DOCKER FOR ENTERPRISE OPERATIONS

HOW WE TEACH

- Docker believes in learning by doing.
- The course is lab driven:
  - Demo Exercises
  - Signature Assignments
- Work together!
- Ask questions at any time

SESSION LOGISTICS

- 2 days duration
- mostly exercises
- regular breaks

ASSUMED KNOWLEDGE AND REQUIREMENTS

- Familiarity with using the Linux command line
- Linux Cheat Sheet: http://bit.ly/2mTQr8l
- A UCP License (download one at https://hub.docker.com/editions/enterprise/docker-ee-trial)
- You should know the basics of Docker and Kubernetes
  - Run a Docker container
  - Set up a service on a Swarm
  - Deploy an app on Kubernetes
  - Search for and pull images from Docker Hub

YOUR LAB ENVIRONMENT

- You have been given several instances for use in exercises.
- Ask instructor for access credentials if you don't have them already.

COURSE LEARNING OBJECTIVES

By the end of this course, learners will be able to:

- Identify the key features of UCP and DTR
- Deploy applications on UCP using Swarm or Kubernetes, governed by secure, role-based authentication and authorization
- Establish a secure supply chain for containerized software development using DTR

INTRODUCTION TO DOCKER ENTERPRISE EDITION

A CONTAINERIZED MINDSET

- Expect churn!
- Talk to your developers
- Abstract your datacenter
- Automate everything.

VMs

Containers

Dog photo jeffreyw; Livestock photo Paul Asman, Jill Lenoble; images CC-BY 2.0

THE SOFTWARE SUPPLY CHAIN

- Image Creation
- Image Distribution
- Container Execution

Docker EE enables security, ease of use, and enterprise-grade control at each of these steps.

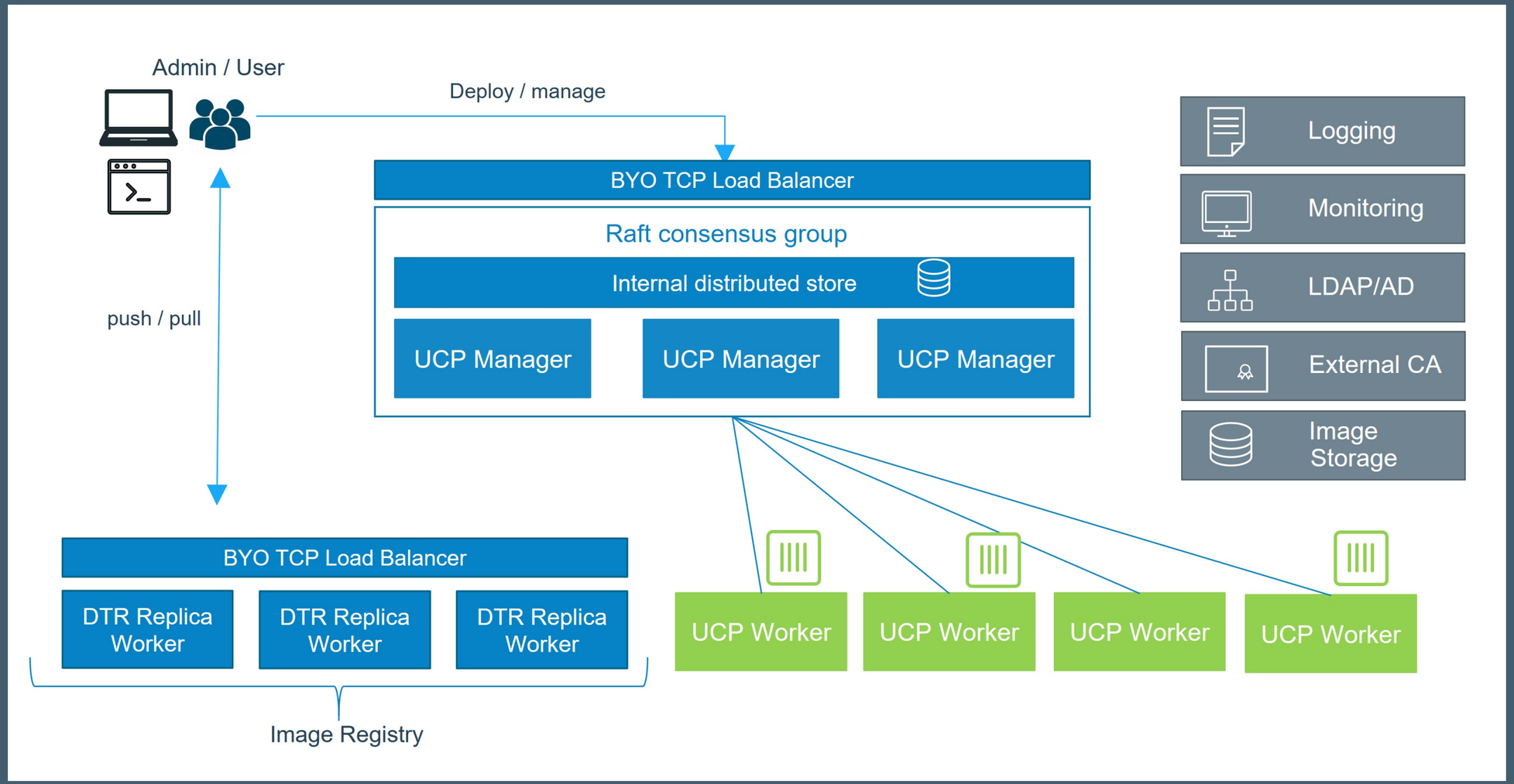
KEY EE FEATURES

- Build:
  - Security Scanning
  - Repository Automation
- Ship:
  - Content Trust
  - Content Cache
- Run:
  - Role based access control
  - Universal Control Plane (GUI)

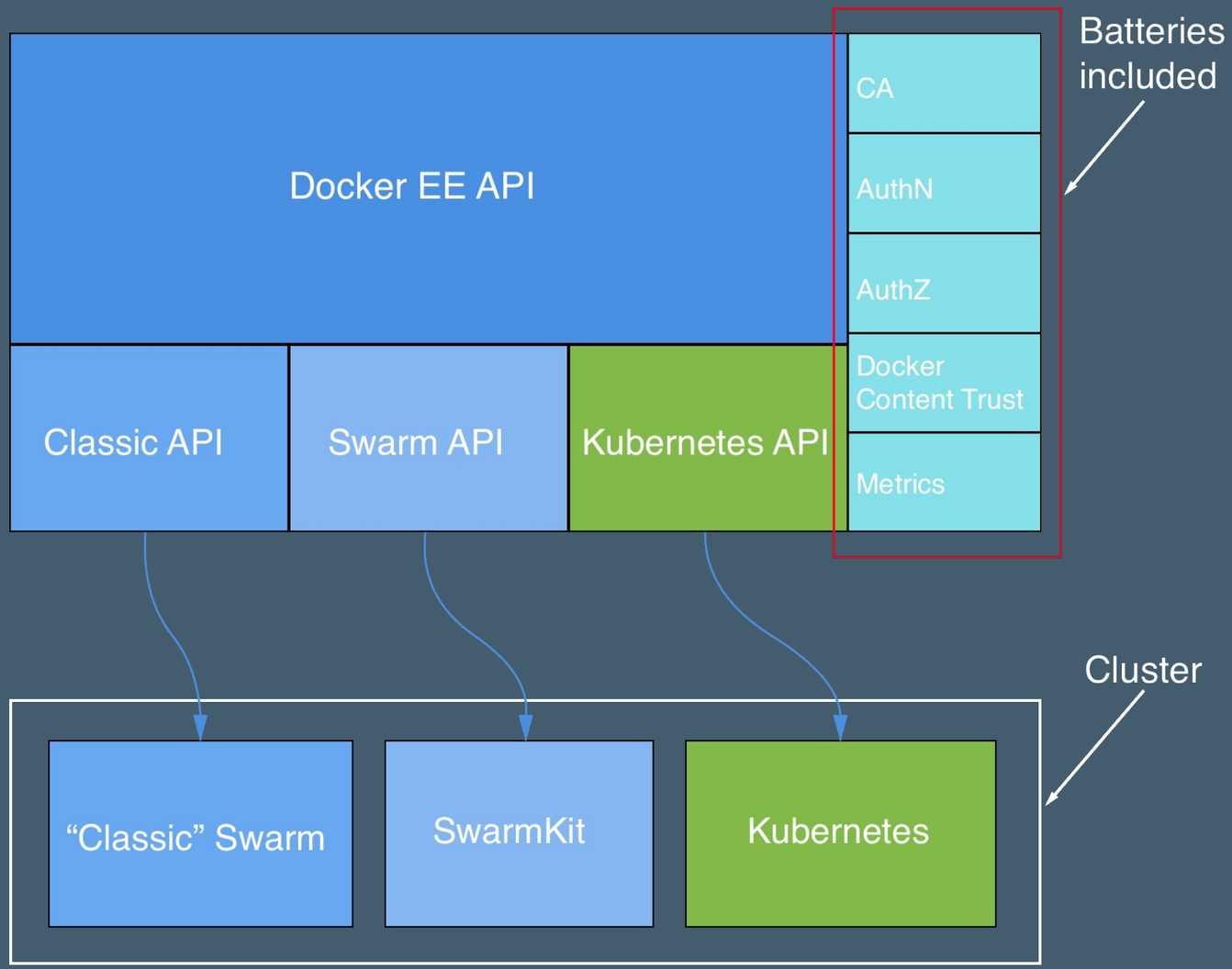
INTEGRATION: BATTERIES INCLUDED BUT SWAPPABLE

- Certificate authorities
- Network drivers
- Storage backends
- User management
- Monitoring
- …

DOCKER EE ARCHITECTURE



CHOOSE YOUR ORCHESTRATOR



EXERCISE: INSTALL UCP

Work through the 'Install UCP' exercise in the Docker for Enterprise Operations Exercises book.

INSTRUCTOR DEMO: CONTAINERIZED NATURE OF UCP

See the 'Containerized Nature of UCP' demo in the Docker for Enterprise Operations Exercises book.

FURTHER READING

- About Docker EE http://dockr.ly/2oq6bPY

UNIVERSAL CONTROL PLANE

LEARNING OBJECTIVES

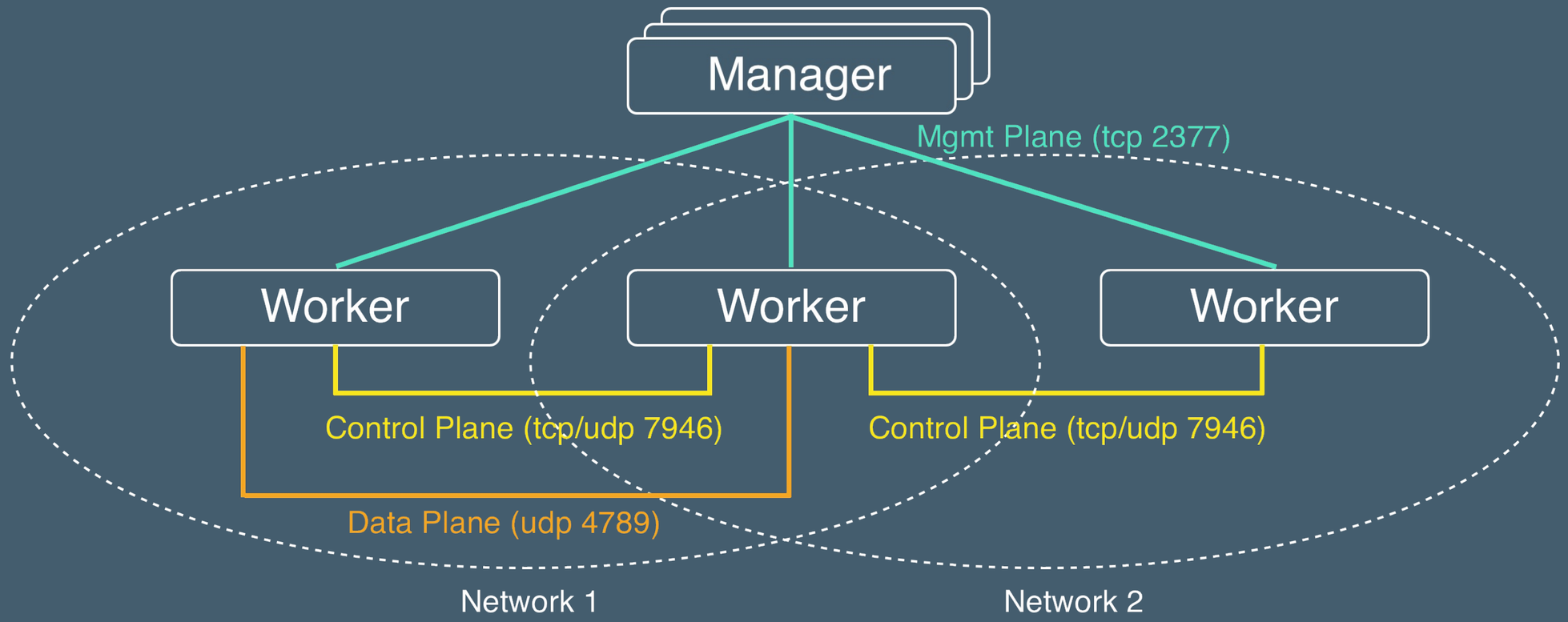
By the end of this module, learners will be able to:

- Describe the interactions and responsibilities of the containers that serve UCP
- Identify the necessary firewall configurations to support inter-node communication
- Create Docker resources by issuing API calls and configuring client bundles

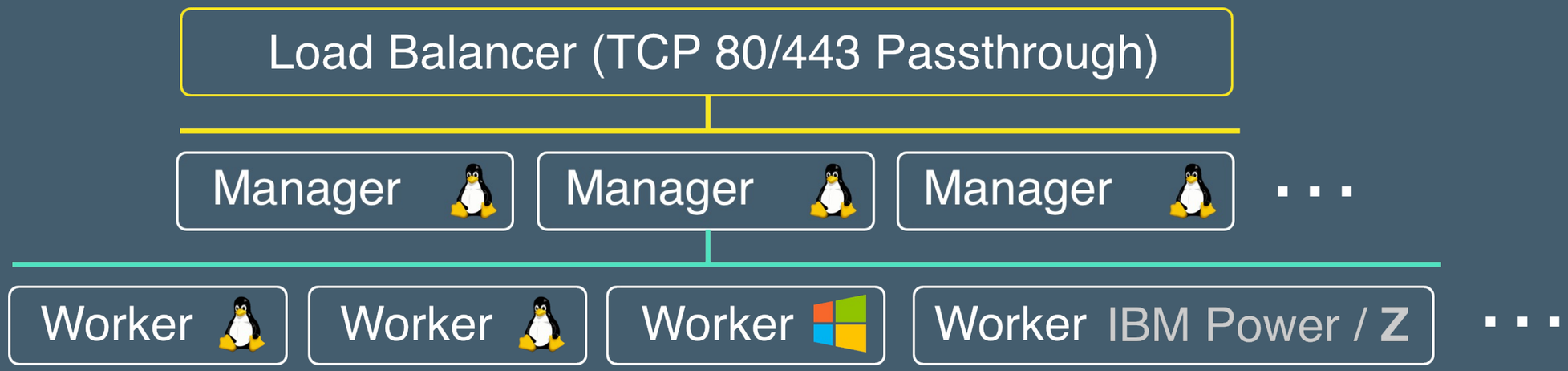
WHAT IS UCP?

- Containerized app running on a Swarm
- Supports Swarm and Kubernetes orchestration
- Adds:
  - Role based access control
  - Secure remote API and CLI access
  - Bootstraps Kubernetes
  - L7 routing

SWARM MODE ARCHITECTURE

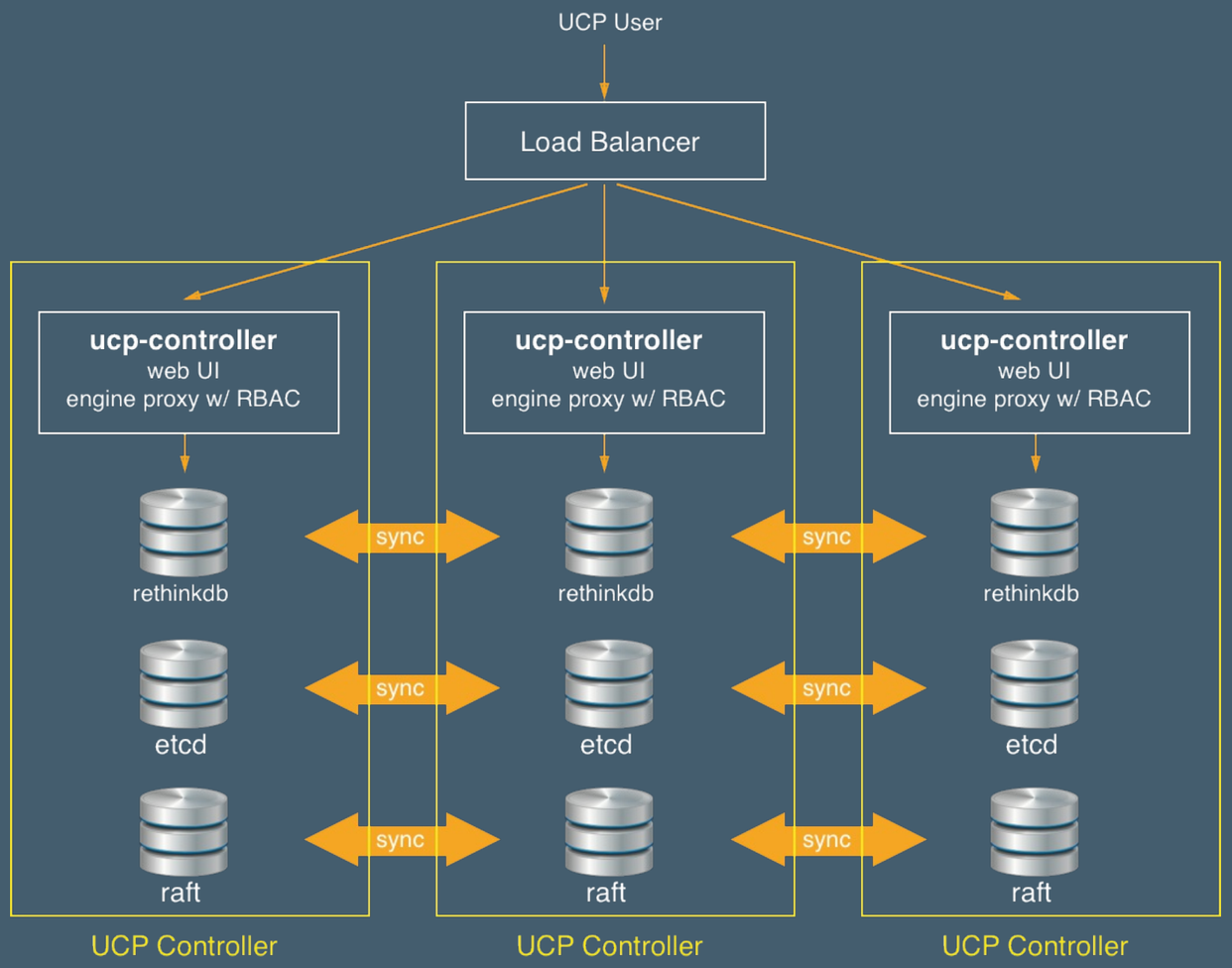


CLUSTER CONFIGURATION

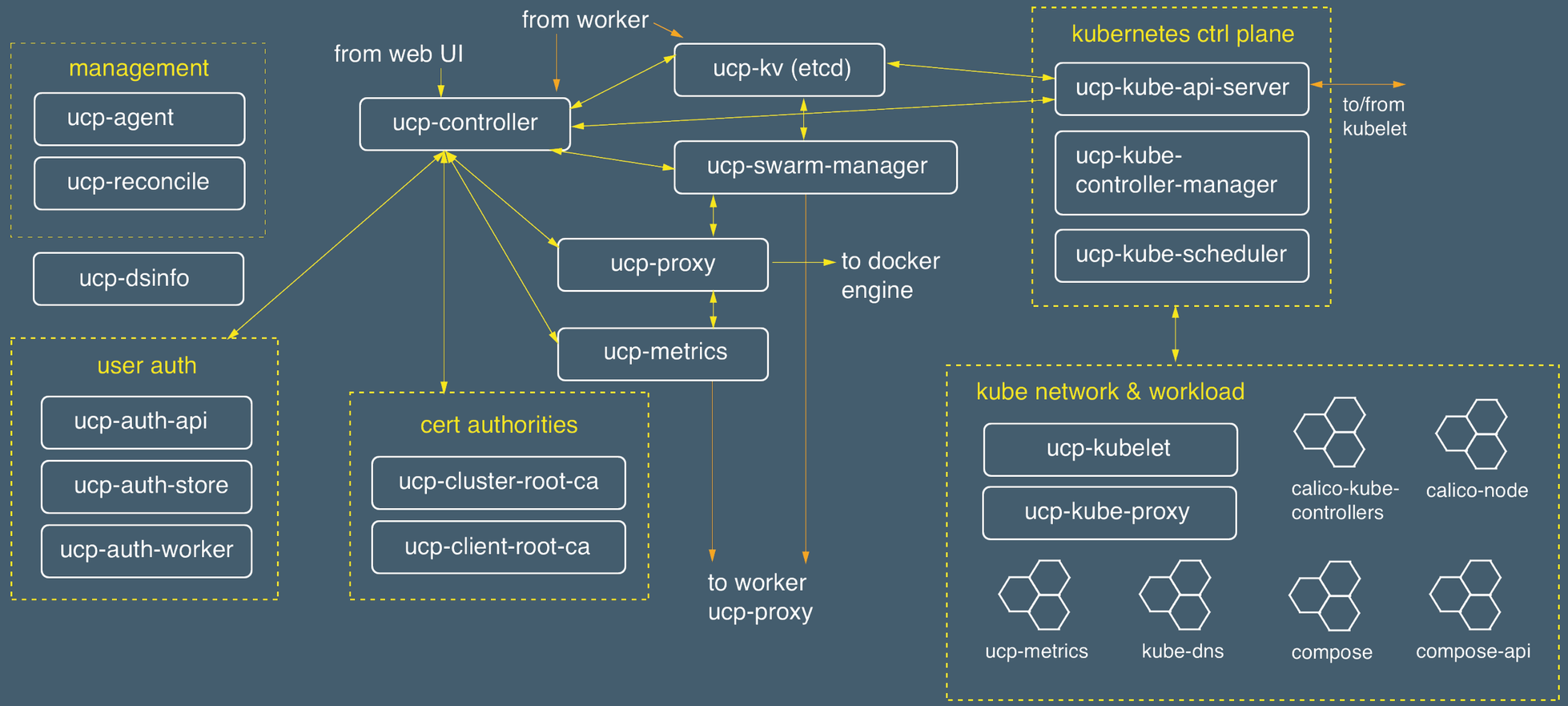


- Odd no. of managers
- Don't run workload on managers
- Don't terminate HTTPS in manager LB
- See /_ping for manager health
- See system requirements at https://dockr.ly/2yBGmlV

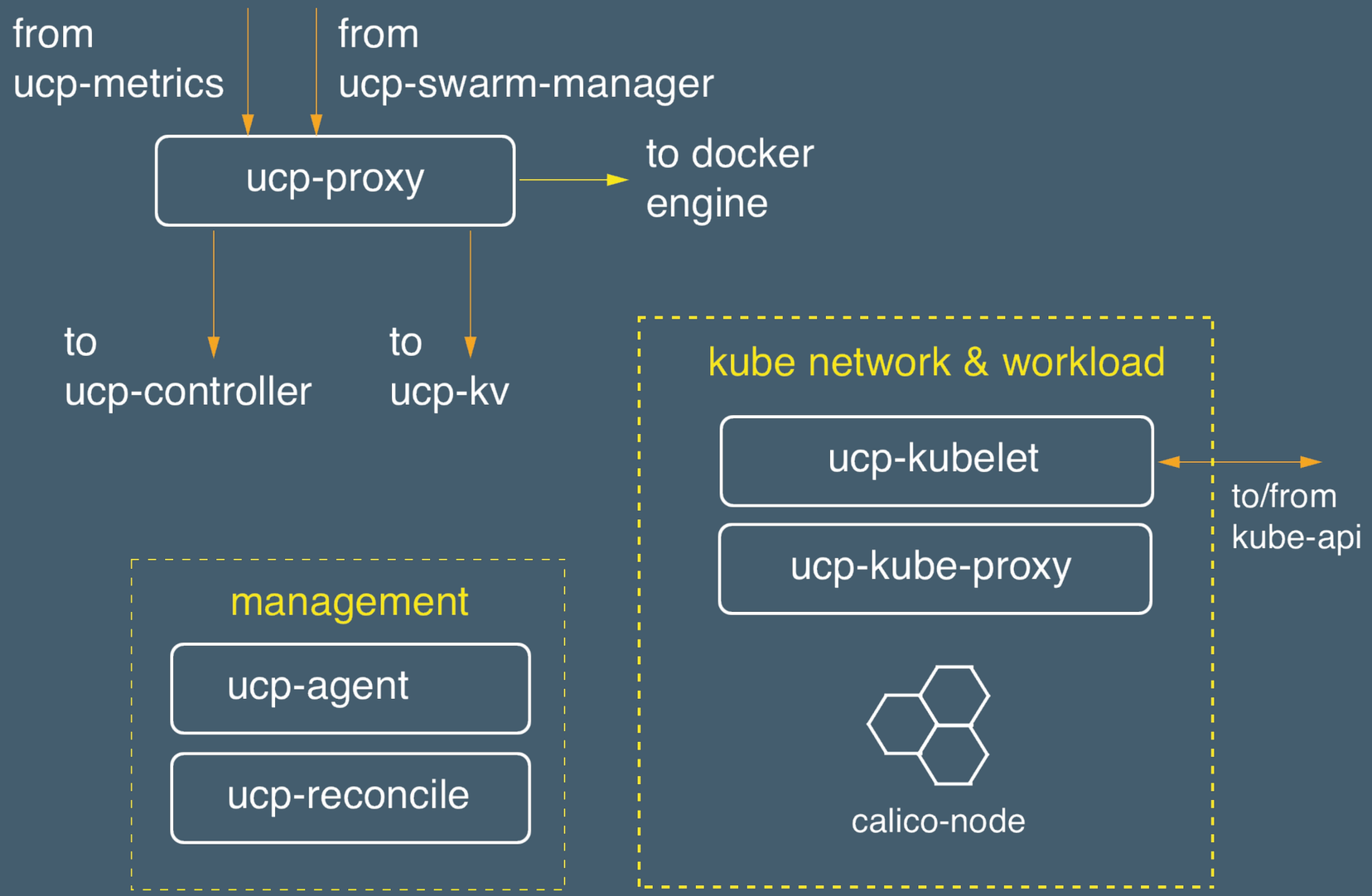
CLUSTER SYNCHRONIZATION



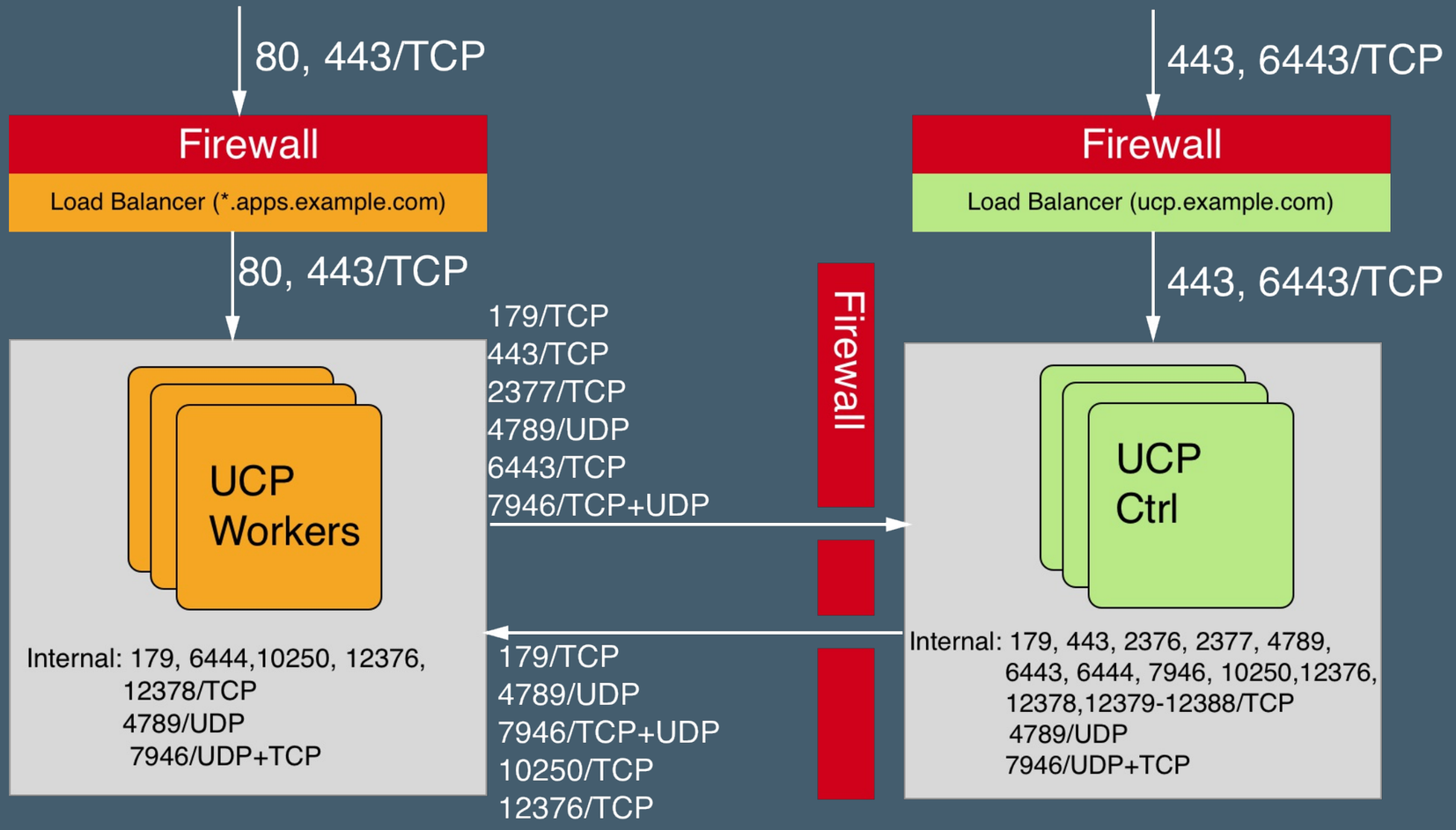
UCP MANAGER ARCHITECTURE OVERVIEW



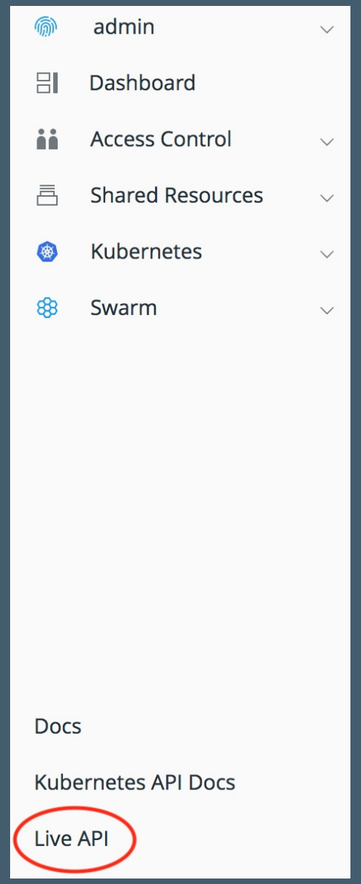
UCP WORKER ARCHITECTURE OVERVIEW



NETWORK TOPOLOGY SUMMARY



UCP API



- Automate UCP via API
- Also used by Web UI
- Docs: https://<UCP FQDN>/apidocs and http://dockr.ly/2E0Hrp1

UCP CLIENT BUNDLES

- Control remote UCP through local Docker and Kubernetes CLI
- Secured with TLS and RBAC
- Certs available in UCP:
  - Web: username -> My Profile -> Client Bundles
  - API endpoint:

```
/api/clientbundle
```

EXERCISE: UCP HIGH AVAILABILITY & ACCESS

Work through:

- (Optional) Configuring UCP for High Availability
- UCP API & Client Bundles

exercises in the Docker for Enterprise Operations Exercises book.

DISCUSSION

- How many managers can you lose and still be able to schedule services, and how many can you lose and still be able to recover your cluster?
- Questions?

FURTHER READING

- Intro to UCP: https://dockr.ly/2K4aXNi
- Docker Reference Architecture: UCP Service Discovery and Load Balancing http://dockr.ly/2rbxDDX
- Docker Reference Architecture: Running Docker Enterprise Edition at Scale: http://dockr.ly/2DW9R3n

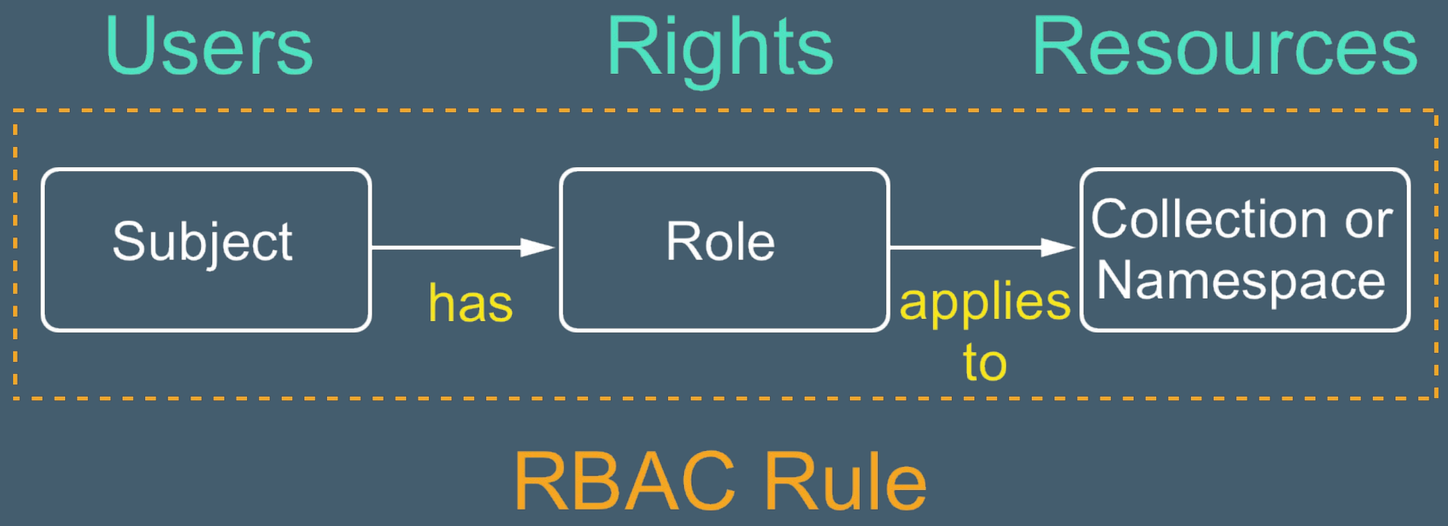
USER MANAGEMENT & ACCESS CONTROL

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Design and write access control rules via UCP for resources managed by both Swarm and Kubernetes

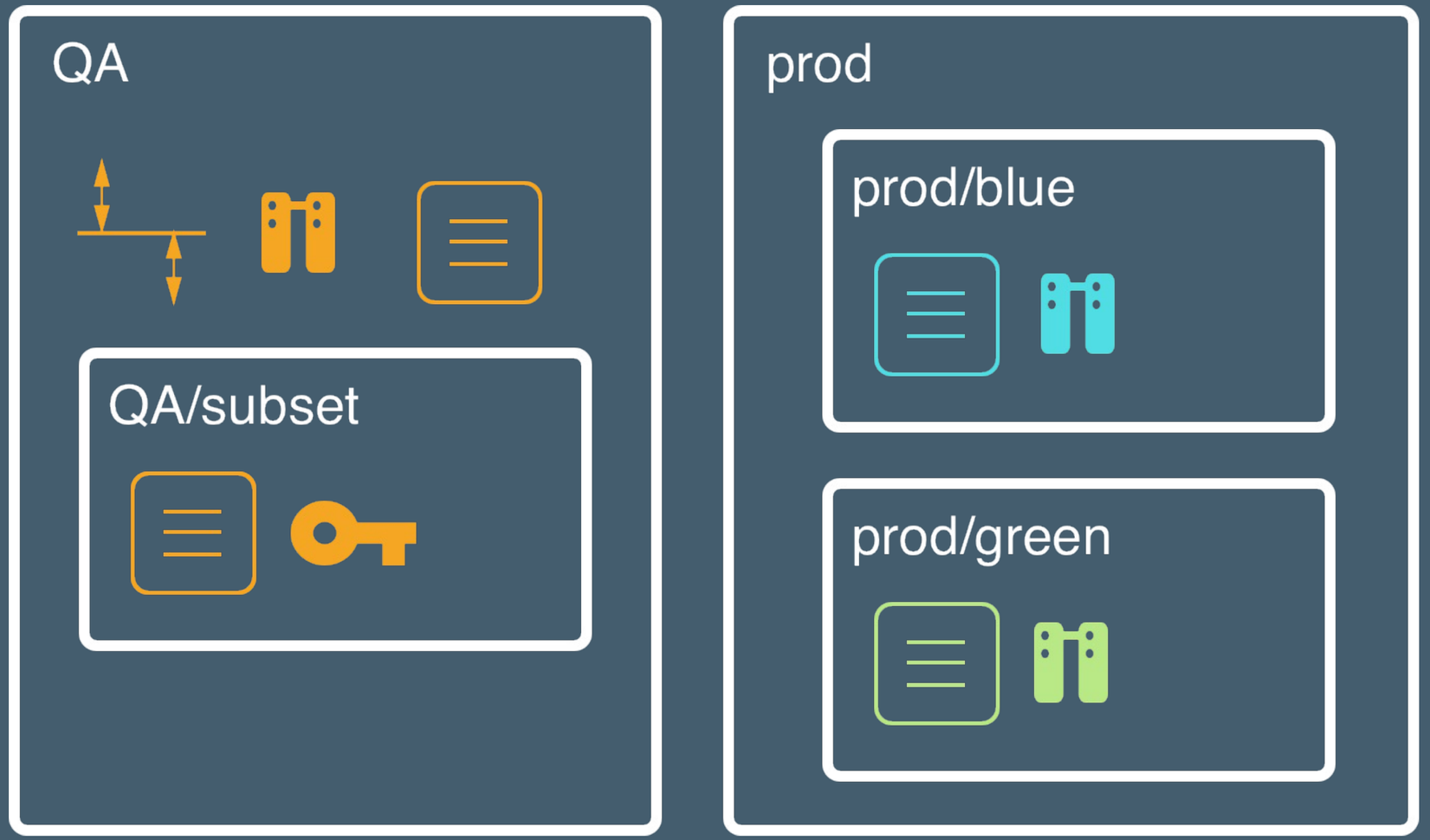
UCP ROLE BASED ACCESS CONTROL



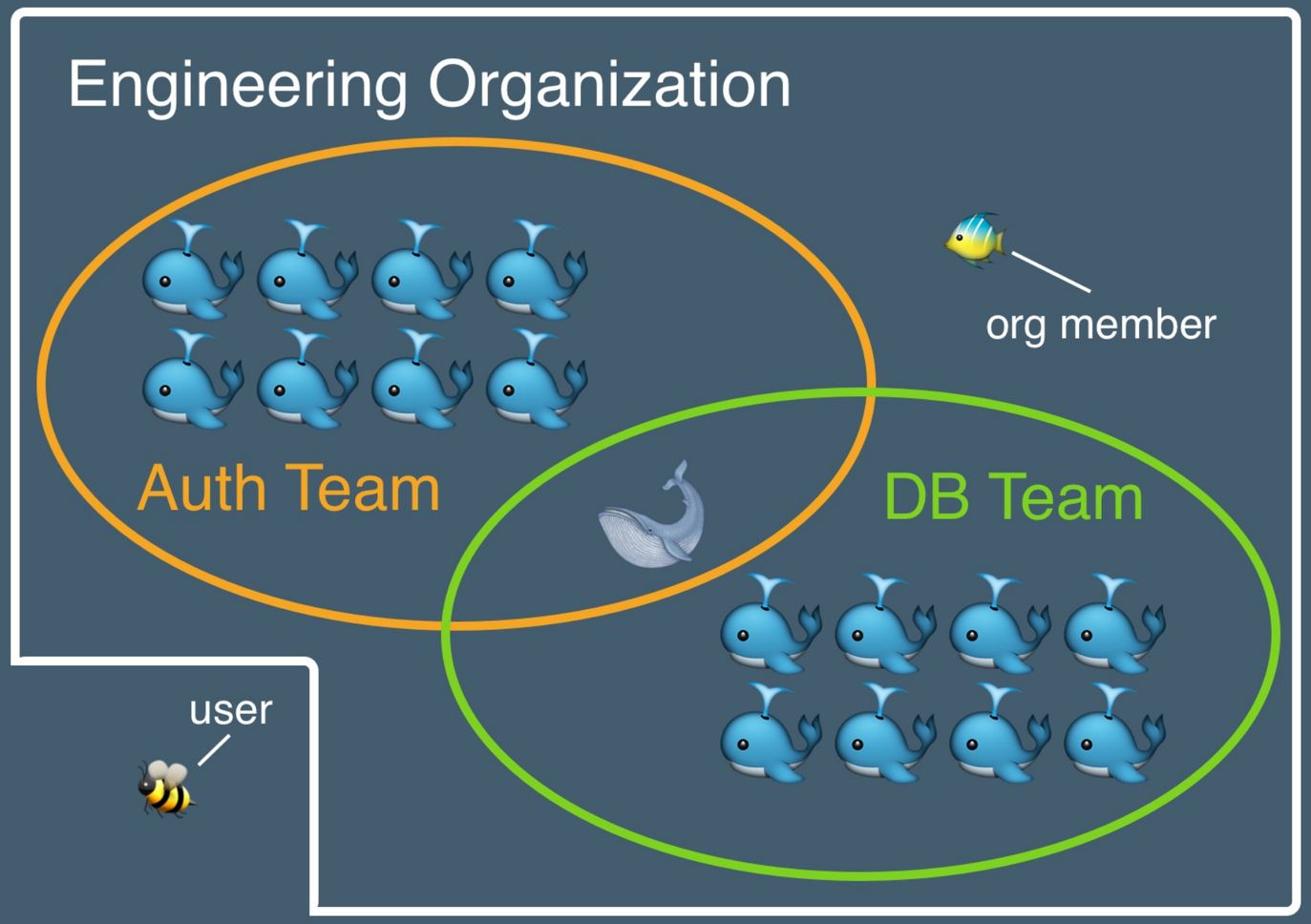
INSTRUCTOR DEMO: RBAC PT. 1

See part 1 of the 'UCP RBAC' demo in the Docker for Enterprise Operations Exercises book.

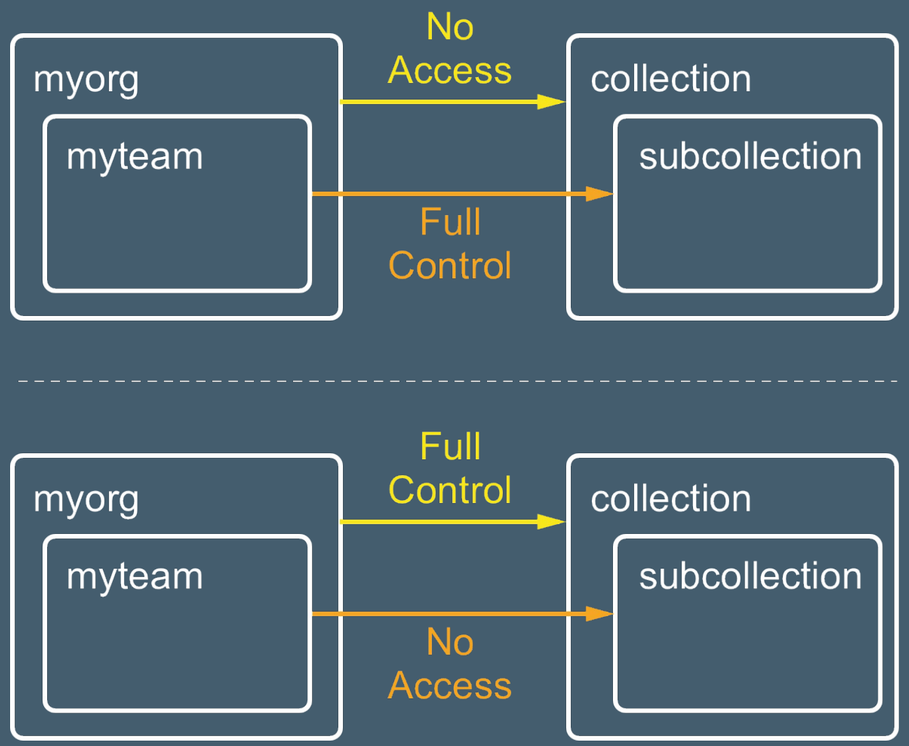
SWARM RESOURCE COLLECTIONS



DOCKER EE SUBJECTS



RBAC CONFLICT RESOLUTION



'myteam' has Full Control of 'subcollection' in BOTH cases!

INSTRUCTOR DEMO: RBAC PT. 2

See part 2 of the 'UCP RBAC' demo in the Docker for Enterprise Operations Exercises book.

PRE-MADE COLLECTIONS

- /System: UCP managers, DTR nodes, system services
- /Shared: worker nodes
- /Shared/Private/<username>: user-specific collections (JIT-provisioned on first login)

DEFAULT COLLECTIONS

- User can have a Default Collection defined
- Default location for new objects created by this user
- Initialized as Collections/Shared/Private/<username>

DEFAULT SWARM ROLES

FULL CONTROL:

- Exec
- Namespaces
- Custom Kernel Capabilities
- Host Bind Mounts
- Privileged Mode

No Node Permissions

RESTRICTED CONTROL:

- Create
- Run
- Restart
- Stop
- Delete

No Node Permissions

VIEW ONLY:

- Inspect
- View

No Node Permissions

NONE

SCHEDULER

- Node Schedule
- Node View

Node permissions

DEFAULT SWARM GRANTS

- Everyone gets 'Restricted Control' to their default collection
- Everyone gets 'Scheduler' to /Shared (where worker nodes are found by default)
- Admins get 'Full Control' to all resource collections

RBAC BEST PRACTICE

Good RBAC is easy to audit, and survives personnel and project changes.

- Almost no one needs full control
- Avoid grants to individual users
- Avoid deep resource collection nesting
- Top level grants should be highly restrictive

RBAC ADVANCED FEATURES

- Custom Swarm roles
- LDAP & SAML integration
- RBAC for node scheduling

CUSTOM SWARM ROLES

- Admins can create new custom roles
- Once created, roles are immutable

LDAP INTEGRATION

- Delegate authentication to an LDAP server
- User accounts will be synced from LDAP based on an LDAP search configuration
- Default just-in-time provisioning
- Must define mapping between LDAP groups & UCP teams

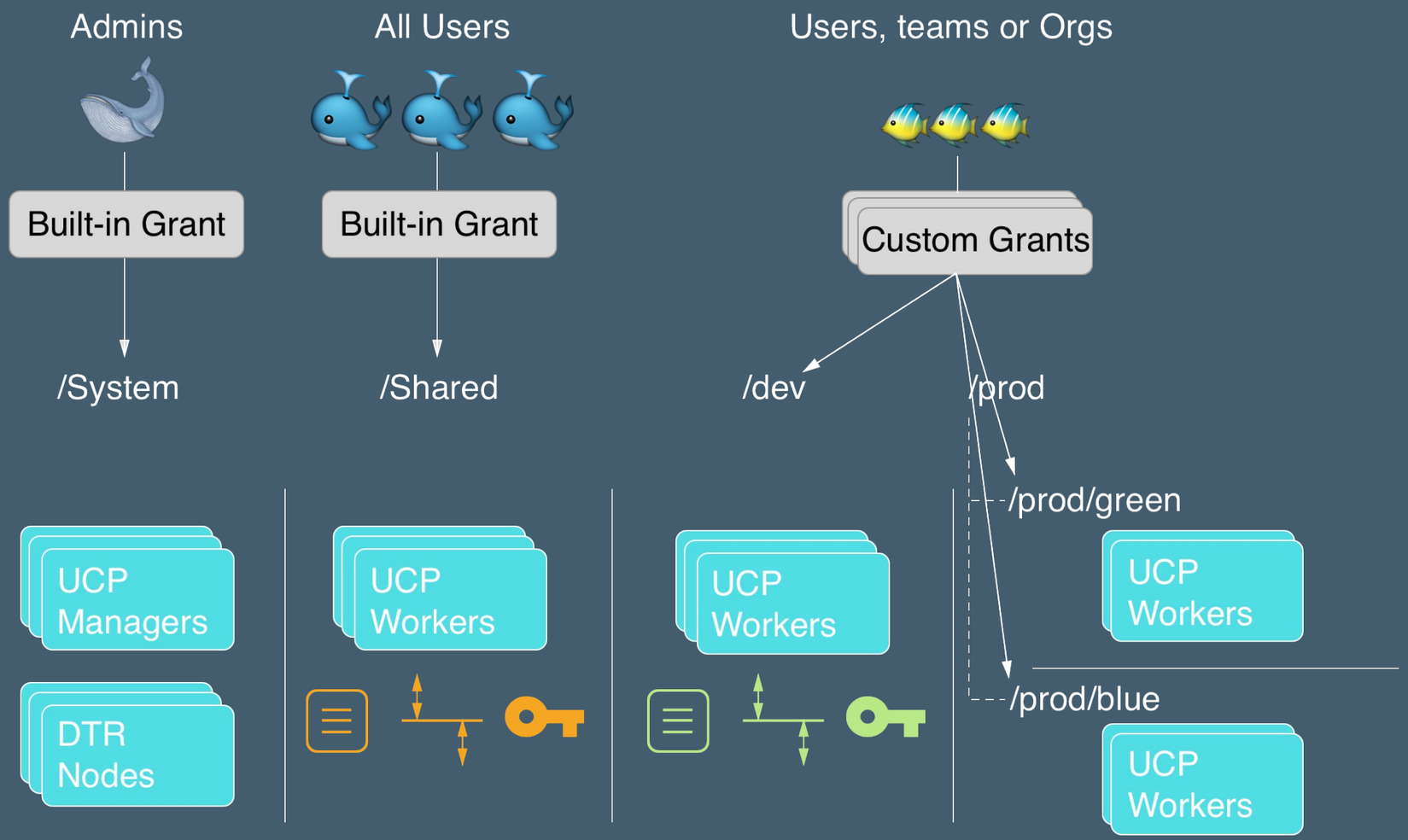
SAML V2.0

- Allows for SSO to UCP through an existing identity provider
- Achieves 2FA through identity provider
- Support for Okta and ADFS, with more identity providers added in the future.

NODE RBAC: STANDARD TIER

- All worker nodes part of /Shared collection
- All users have Scheduler grant to /Shared collection

NODE RBAC: ADVANCED TIER



KUBERNETES RBAC IN UCP

- Shared users, teams and organizations
- Namespaces instead of resource collections
- Roles and ClusterRoles instead of Swarm roles
- RoleBindings instead of grants

KUBERNETES NAMESPACES

Auto-generated

default

- default namespace for user content

kube-system

- calico
- kube-dns
- compose adapter

kube-public

User-defined

development

prod-east

prod-west

KUBE ROLES & CLUSTERROLES

- Conceptually equivalent to Swarm roles
- 'Roles' scoped to a single Namespace
- 'ClusterRoles' usable across or above namespaces

DEFAULT KUBE CLUSTERROLES

- cluster-admin: Cluster-wide admins
- admin: Single-ns admin
- edit: r/w most objects in ns
- view: r/o
- 'system:x': system resources, don't touch

CUSTOM KUBE ROLES

- Can be defined for any API group and resource
- Specify a list of verbs allowed

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-development
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

KUBE ROLEBINDINGS

- Equivalent to a Swarm Grant
- Combine a subject, namespace and role or clusterRole
- Defaults:
  - No access for users
  - Admins get cluster-admin rights cluster-wide

NODE RBAC: KUBERNETES

1. Put nodes in a Swarm collection
2. Associate collection with namespace
3. Result: resource in namespace will only be scheduled on associated nodes.

INSTRUCTOR DEMO: RBAC PT. 3

See part 3 of the 'UCP RBAC' demo in the Docker for Enterprise Operations Exercises book.

EXERCISE: USER MANAGEMENT

Work through:

- Access Control in UCP
- User Management with LDAP
- Password Recovery

in the Docker for Enterprise Operations Exercises book.

DISCUSSION

- What are some pros and cons of RBAC for worker nodes, compared to having separate clusters?
- Questions?

FURTHER READING

- Access control model: https://dockr.ly/2HhTiUN
- Create users and team manually: https://dockr.ly/2HgZdJo
- Create teams with LDAP: https://dockr.ly/2HTAtnX
- Integrate with an LDAP directory: https://dockr.ly/2K3jC2z
- Kubernetes RBAC: https://bit.ly/2yDGEZB

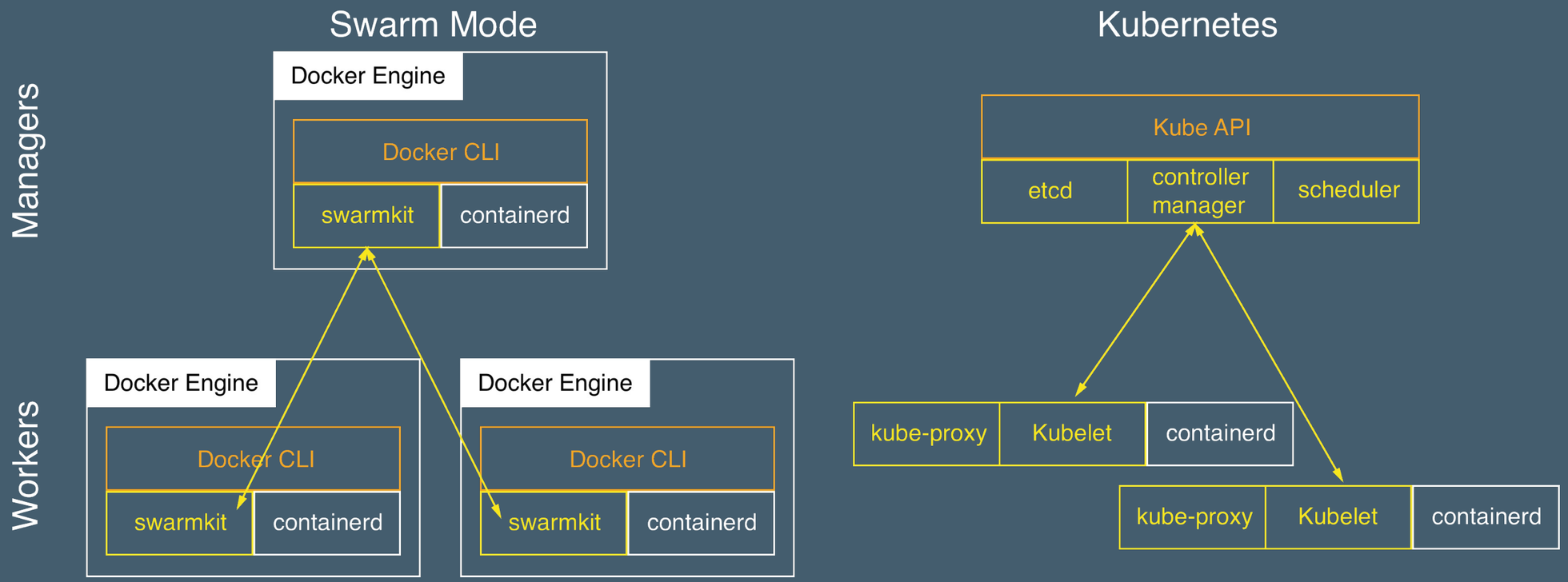
UCP ORCHESTRATION

LEARNING OBJECTIVES

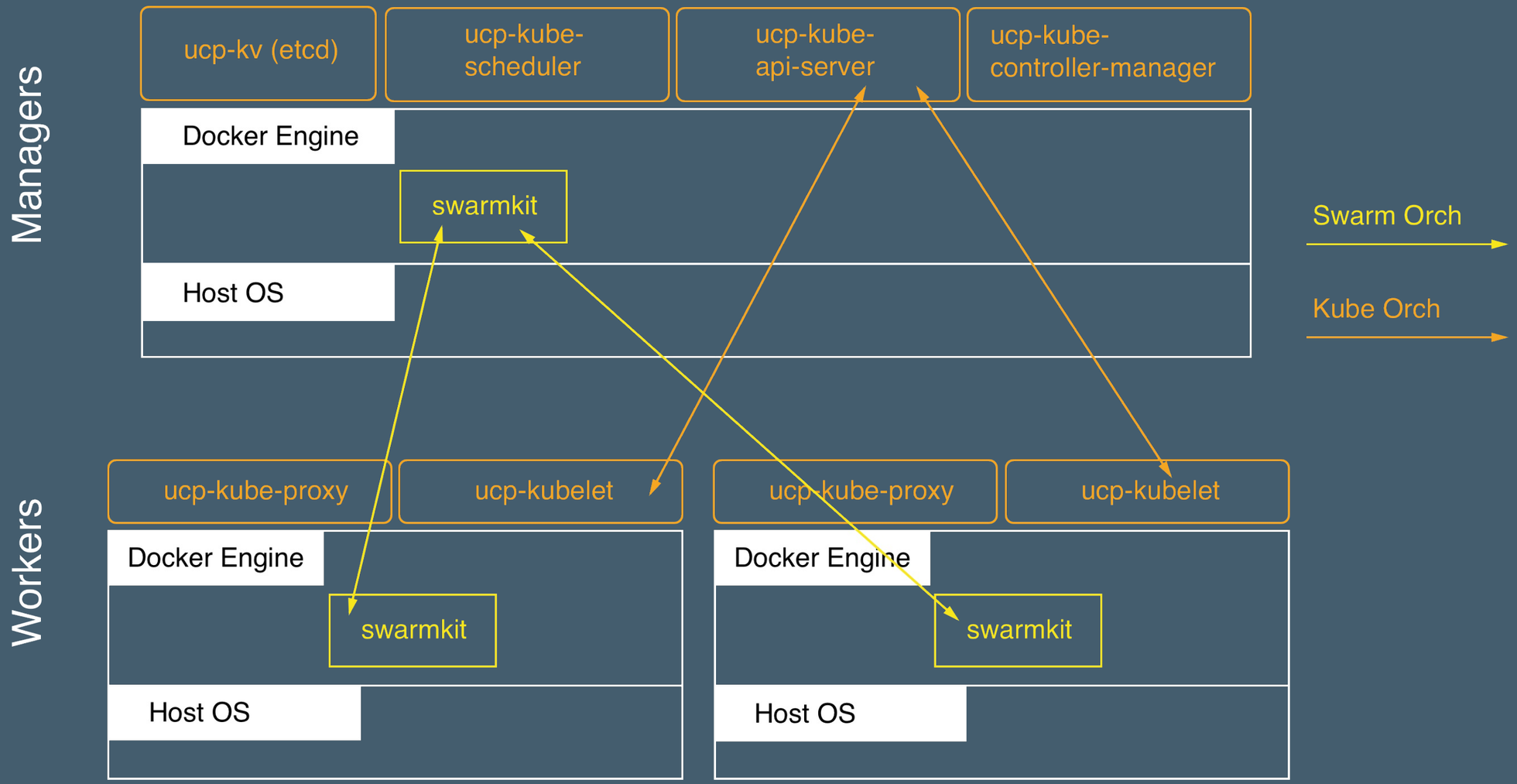
By the end of this module, learners will be able to:

- Compare and contrast Swarm and Kubernetes orchestration components and networking models
- Use UCP to deploy apps on both orchestrators

ORCHESTRATOR ARCHITECTURE



ORCHESTRATORS IN UCP



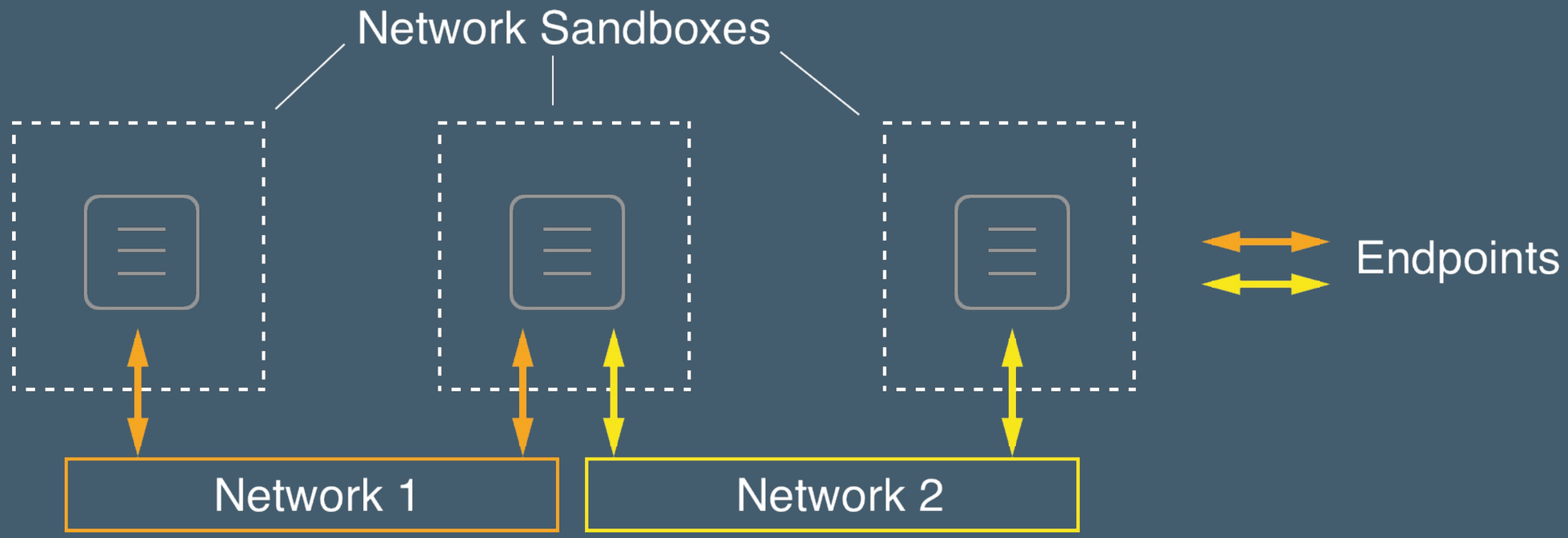
ADDITIONAL KUBERNETES COMPONENTS

- Kube DNS
- Calico (default)
  - Install UCP with custom CNI: `--cni-installer-url`
  - No CNI on UCP install: `--unmanaged-cni`
- Compose Adapter

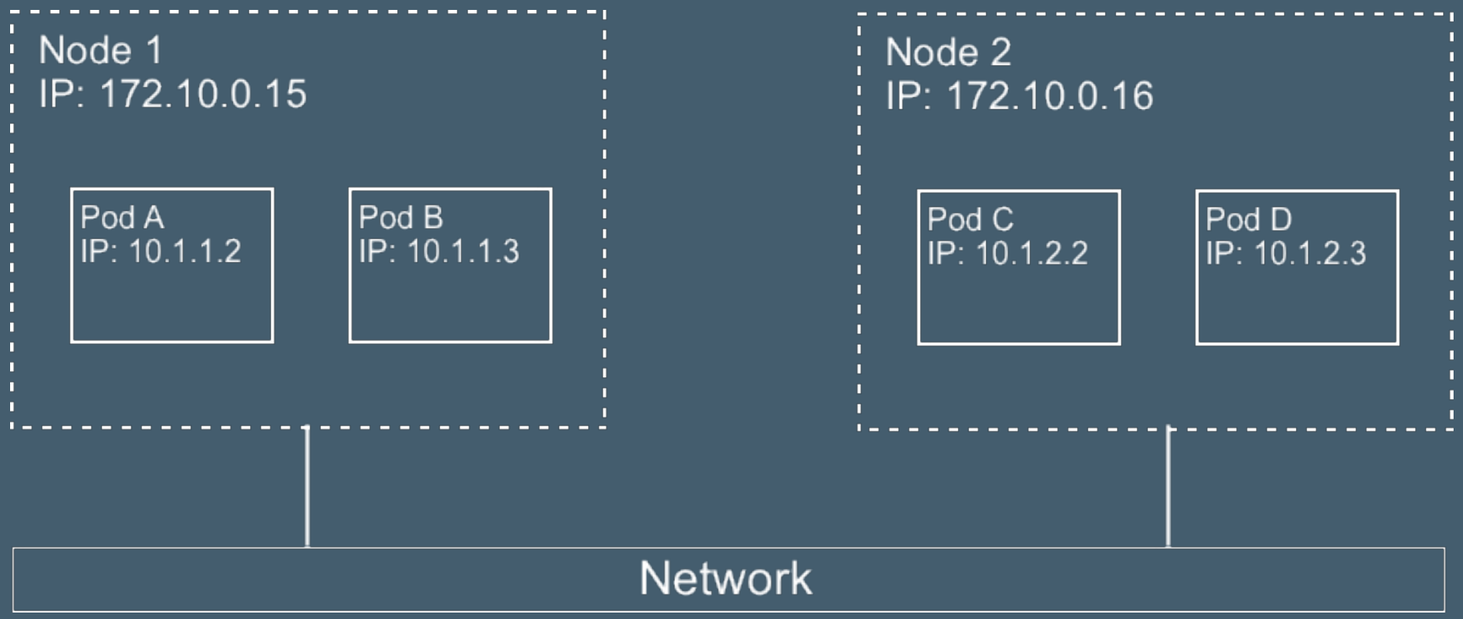
NETWORKING MODELS

- Fundamental spec for how containers communicate
- Flexible and high level
- Standardization for how networks are built
- Distinct for Docker native vs. Kubernetes

DOCKER'S CONTAINER NETWORK MODEL



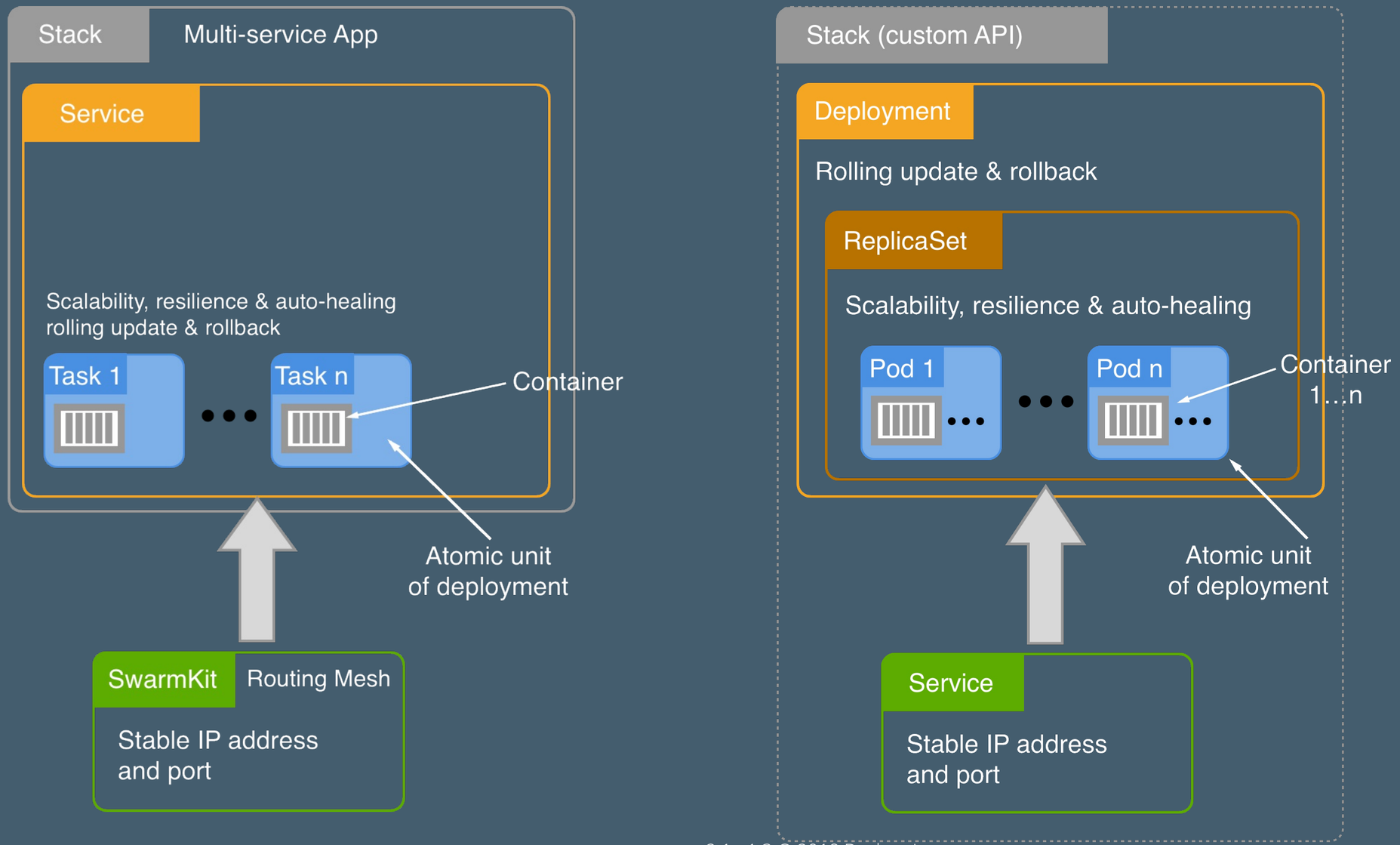
KUBERNETES NETWORK MODEL



Requirements

- Pod <--> Pod without NAT
- Node <--> Pod without NAT
- Pod's peers find it at the same IP it finds itself
- Creates a flat network, like VMs

ORCHESTRATION COMPONENTS



ORCHESTRATOR UI

- Stacks via compose yaml
- Swarm objects via webforms
- Kube objects via kube yaml
- All also reachable via API calls.

DEFINING APPLICATION STACKS

- Deployable to both Swarm and Kubernetes
- Two options:
  - Compose file: deploys stack across cluster
  - App package: superset of compose file; includes metadata, environment configuration

APPLICATION DATA STORAGE

- Kubernetes:
  - NFS
  - AWS (EBS, EFS)
  - Azure (File, Disk)
  - vSphere
- Swarm:
  - Cloudstor (AWS EBS and Azure Disk)
  - NFS via `local` volume driver

EXERCISE: ORCHESTRATING APPLICATIONS IN UCP

Work through:

- Orchestrating Applications
- Combining Collections and Kubernetes Namespaces

in the Docker for Enterprise Operations Exercises book.

FURTHER READING

- Docker & Kubernetes: https://www.docker.com/kubernetes
- Official Kubernetes Docs: https://kubernetes.io/docs
- Kubernetes tutorials: http://bit.ly/K8-tutorials
- Understanding Kubernetes Networking: http://bit.ly/2kdI1qQ
- Kubernetes the Hard Way: http://bit.ly/29Dq4wC
- Installing custom CNI plugins to UCP: https://dockr.ly/2DwJILX

CONTAINER NETWORK OPERATIONS

DISCUSSION: SENDING REQUESTS

Imagine deploying an API as a Swarm service or Kube deployment. What are some networking considerations to make when sending requests to these containers?

LEARNING OBJECTIVES

- Route traffic at L3, L4 and L7 on Swarm or Kubernetes
- Configure load balancing and routing appropriately for stateful and stateless applications
- Configure deployment models such as canary and blue/green

STEPS OF COMMUNICATION

1. Service Discovery / Ingress
2. Load Balancing
3. Routing

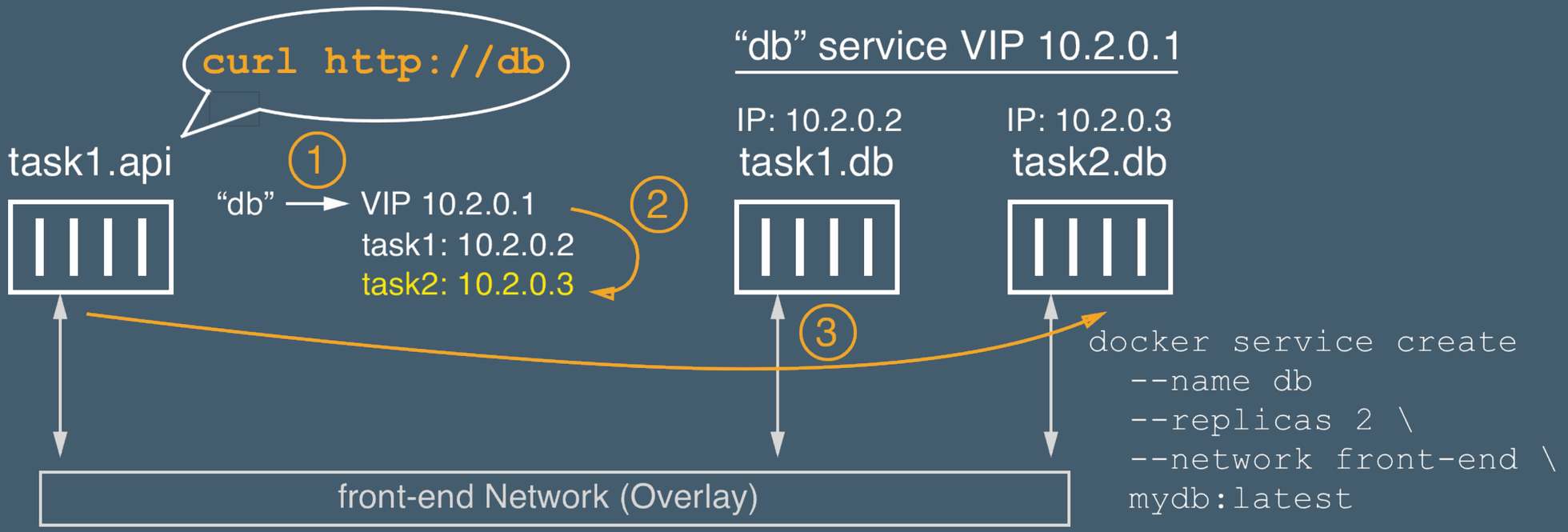
CHOOSING THE RIGHT NETWORKING TOOL

Three questions:

- Is the request originator internal or external to your cluster?
- Are the destination containers stateless or stateful?
- Are you using Swarm or Kubernetes?

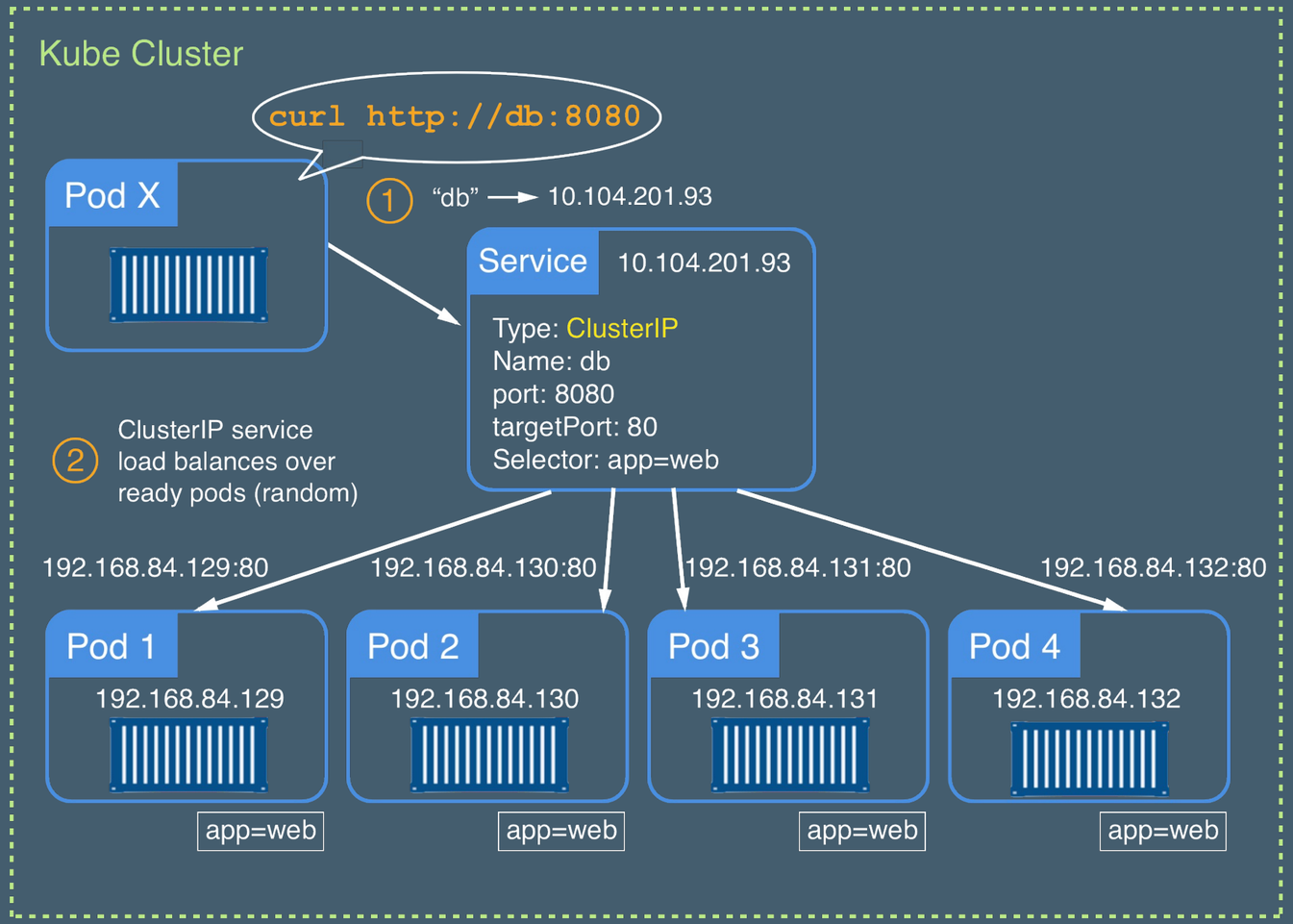
INTERNAL / STATELESS / SWARM

Solution: Swarm VIPs



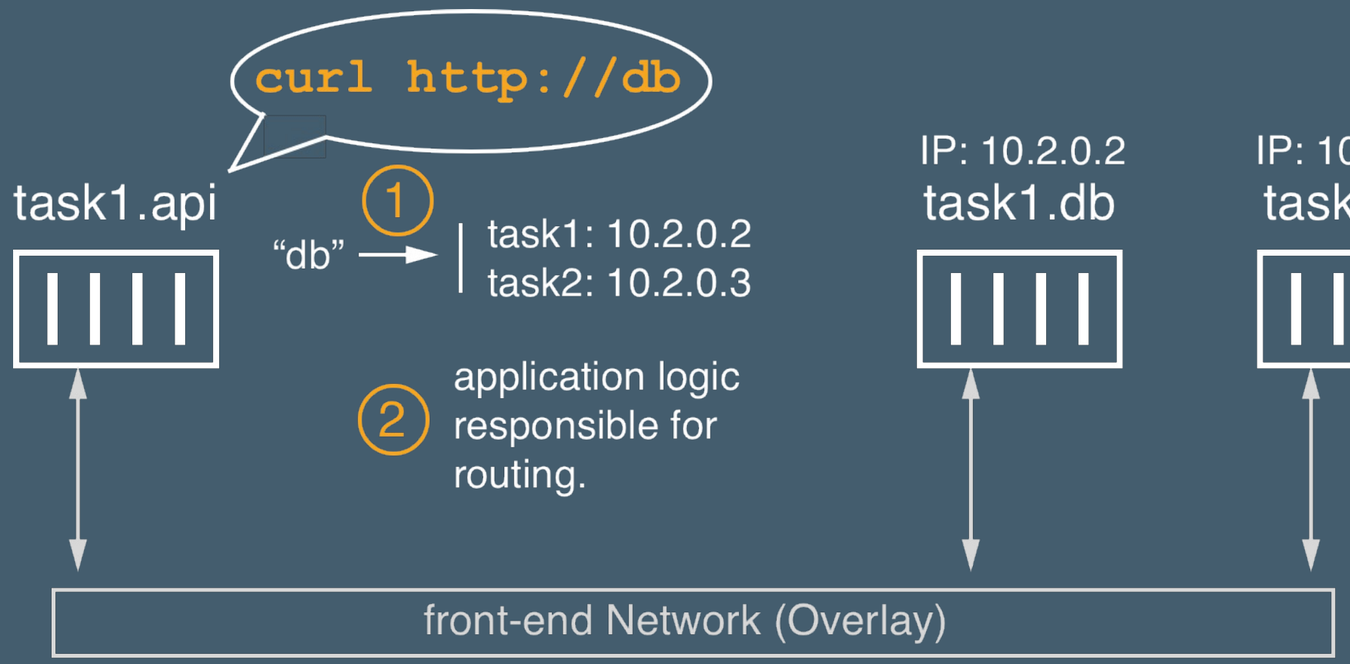
INTERNAL / STATELESS / KUBE

Solution: ClusterIP Service



INTERNAL / STATEFUL / SWARM

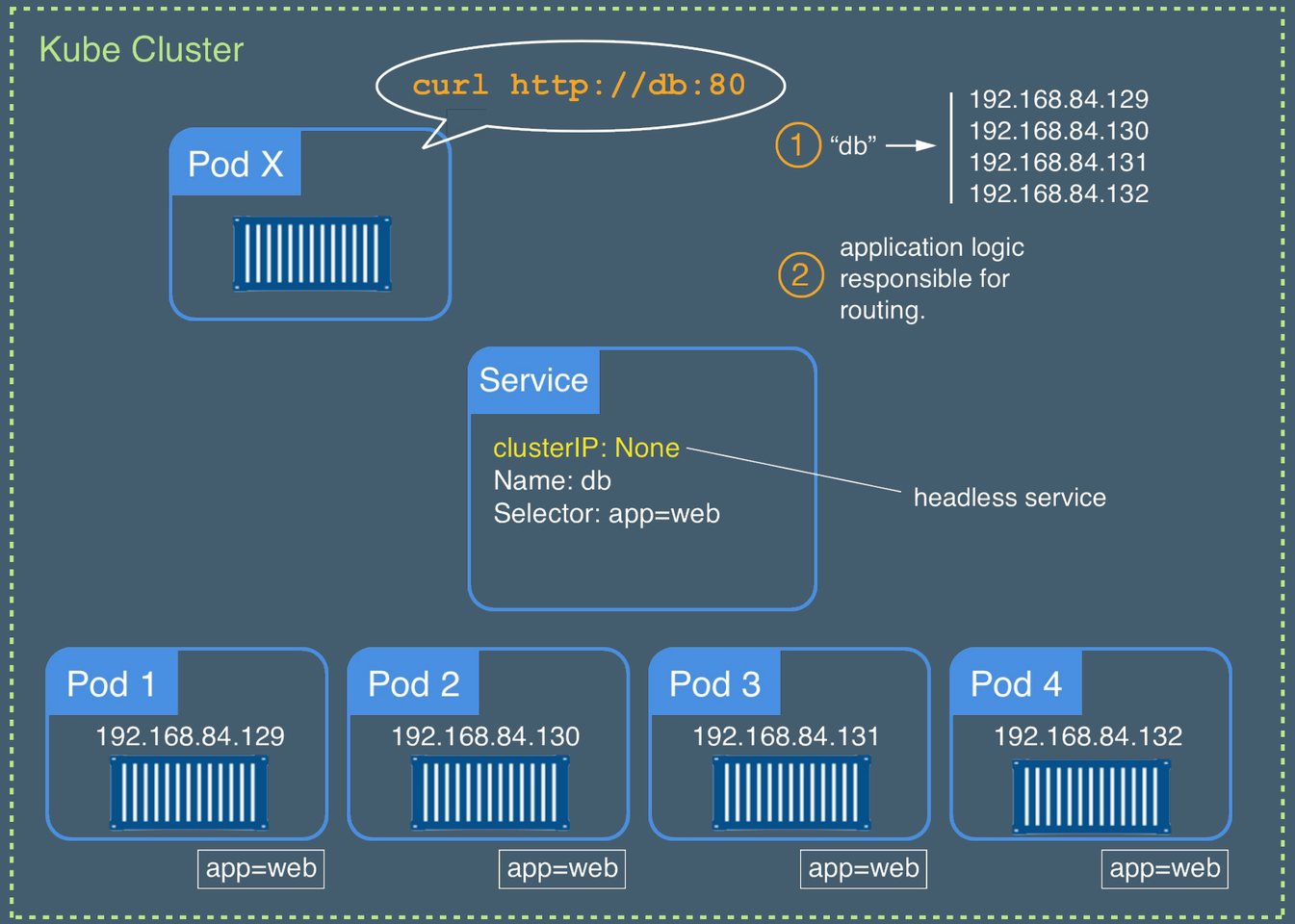
Solution: DNSRR Endpoints




```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  --endpoint-mode dnsrr
  mydb:latest
```

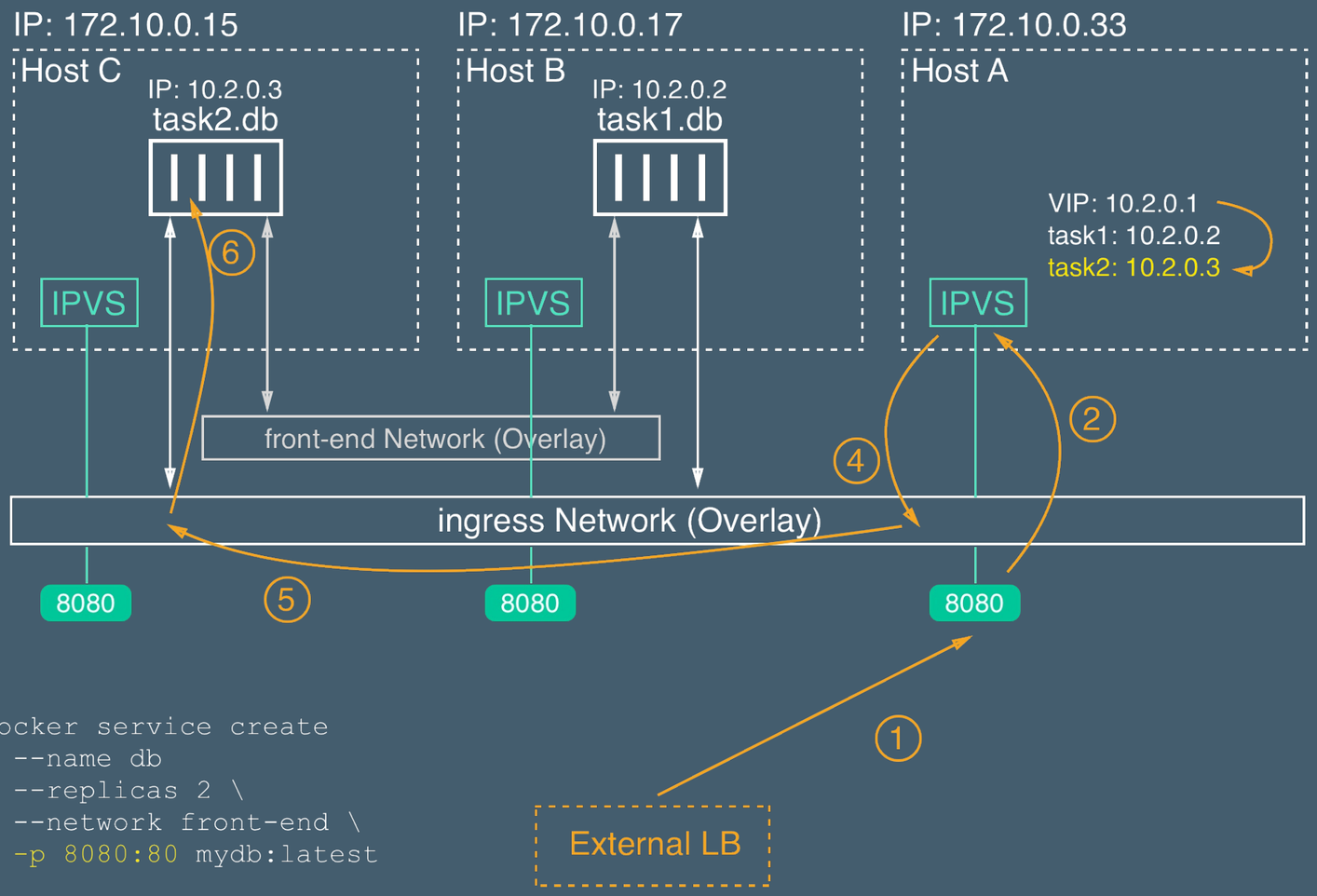
INTERNAL / STATEFUL / KUBE

Solution: Headless ClusterIP



EXTERNAL / STATELESS / SWARM

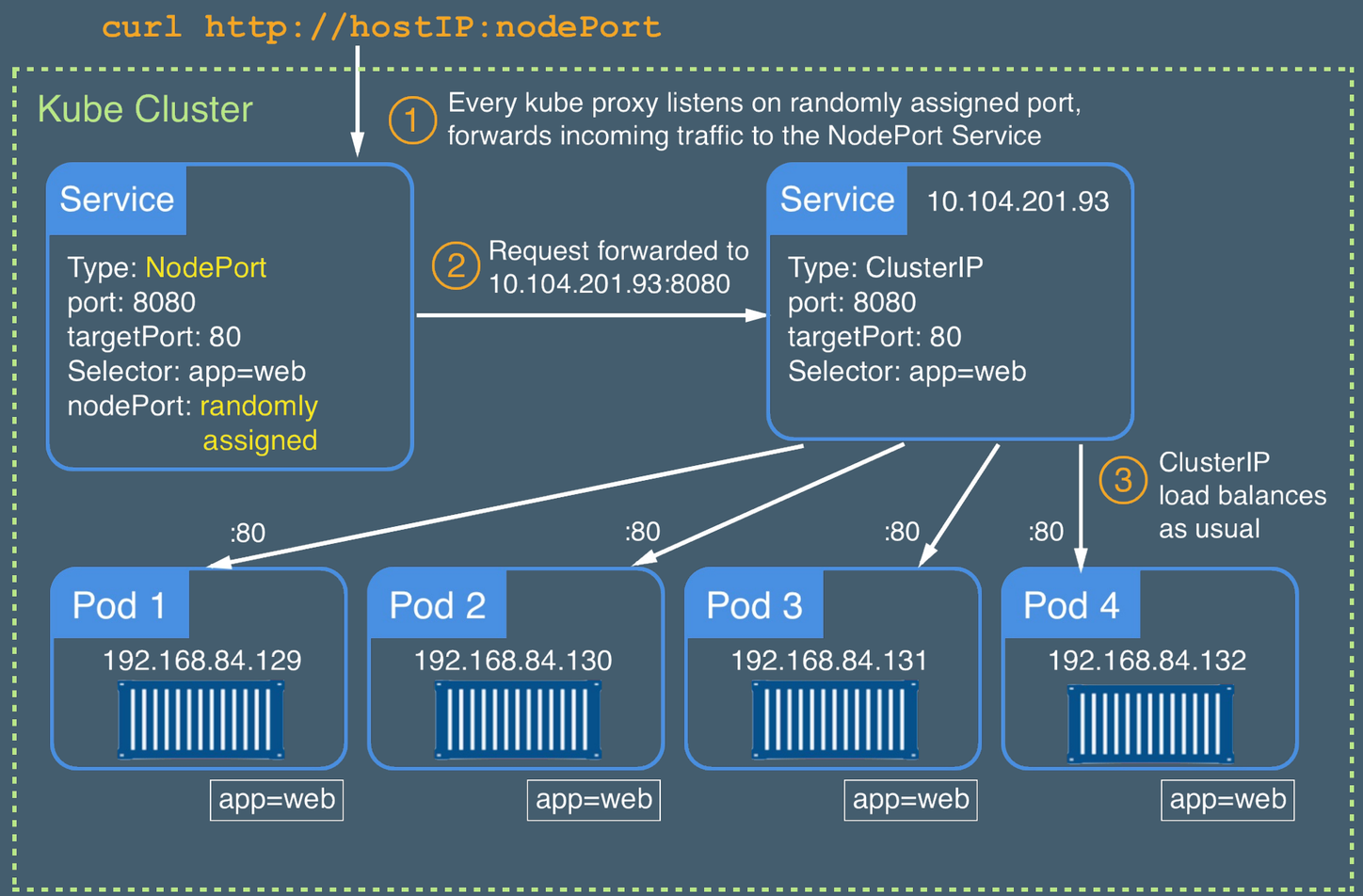
Solution: Swarm L4 Routing Mesh




```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  -p 8080:80 mydb:latest
```

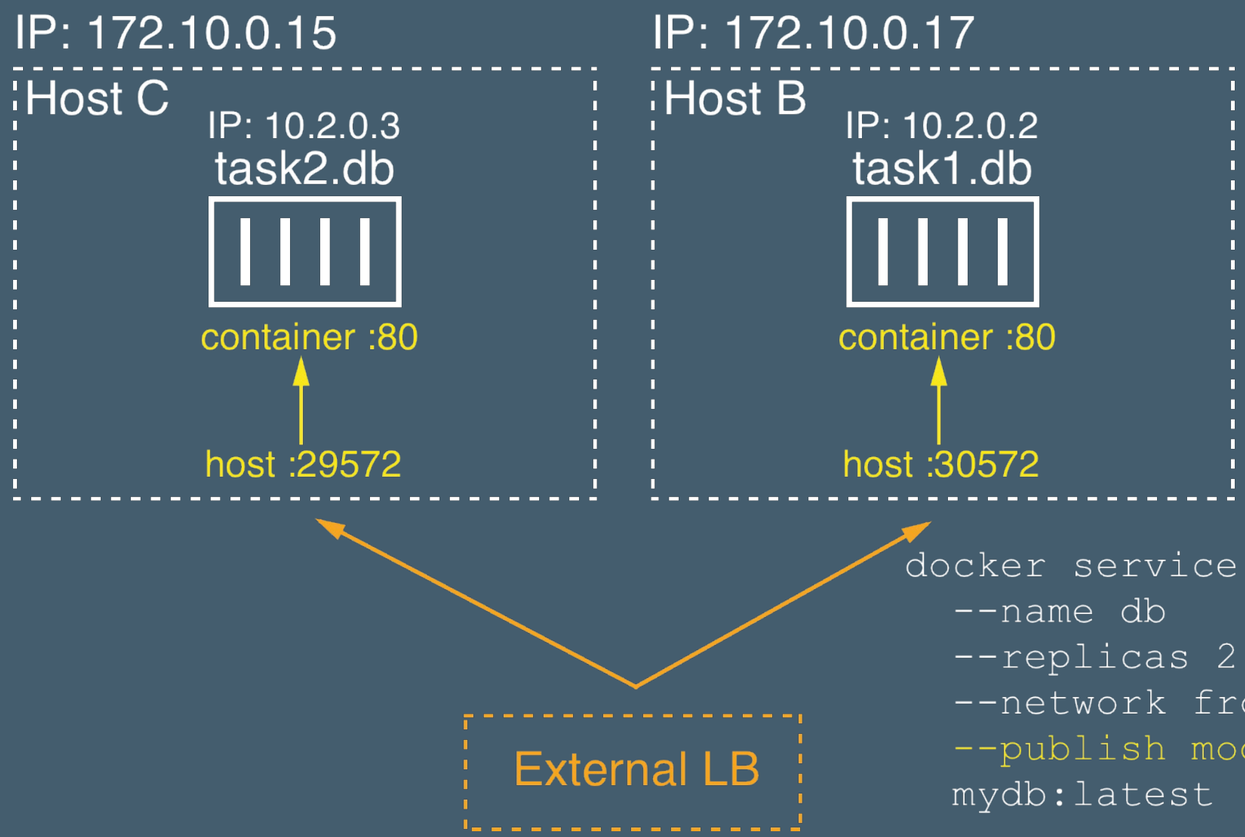
EXTERNAL / STATELESS / KUBE

Solution: NodePort Service



EXTERNAL / STATEFUL / SWARM

Solution: --publish mode=host




```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  --publish mode=host,target=80
  mydb:latest
```

EXTERNAL / STATEFUL / KUBE

- Kube discourages host/pod port mapping at scale: https://bit.ly/2pU0ECr
- Options:
  - **Ingress** with session persistence (see below; allows consistent but not specific connections)
  - **DaemonSet** with a **hostPort** configured per pod (scale limited to size of cluster)
  - One **NodePort** service per pod (impractical for large deployments)

EXERCISE: BASIC ROUTING MODELS

Work through:

- Basic Swarm Routing Models
- Basic Kubernetes Routing Models

in the Docker for Enterprise Operations Exercises book.

ADVANCED OPTIONS

- UCP Interlock 2
  - Swarm L7 routing
  - Sticky sessions
- Kube Ingress
  - Kube L7 routing
  - Path-based routing
  - Sticky Sessions

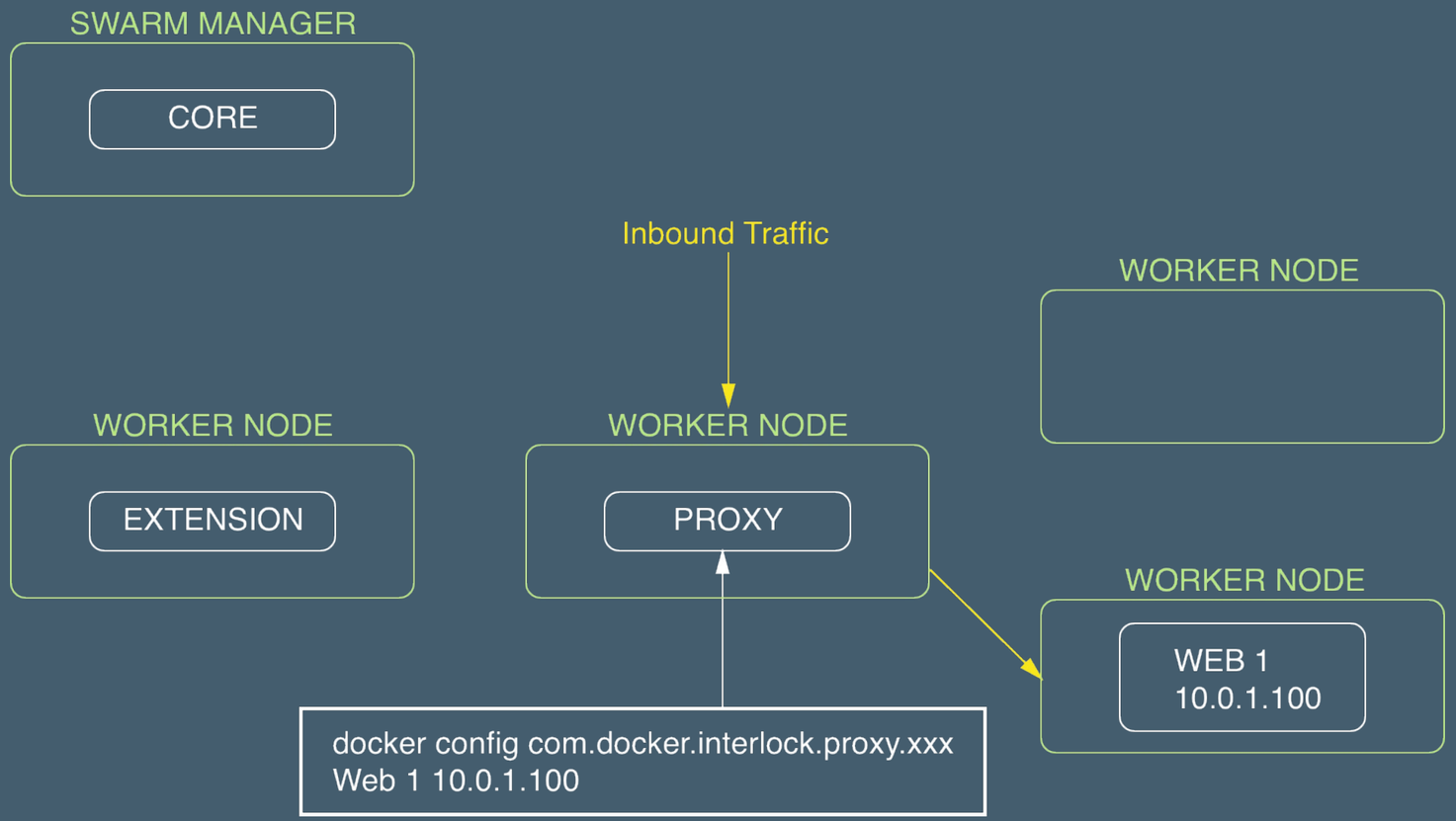
L7 ROUTING

- Route traffic to service based on Host header
- Minimizes external load balancer reconfiguration
- Supported in UCP and Kubernetes

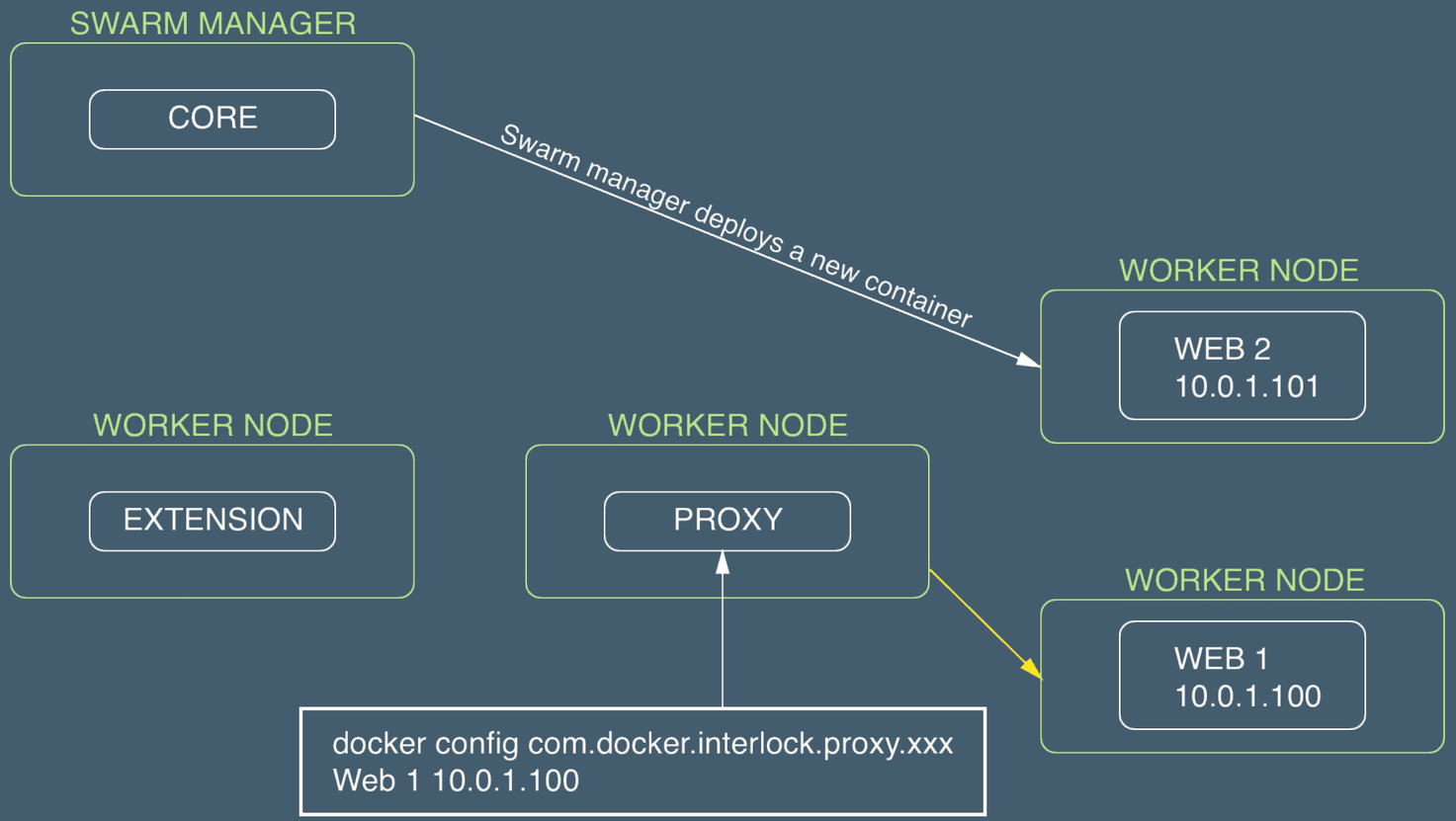
L7 ROUTING - SWARM: INTERLOCK 2

- Core: communicates scheduler decisions to extension service
- Extension: manages proxy config automatically
- Proxy: nginx or haproxy; ingress point

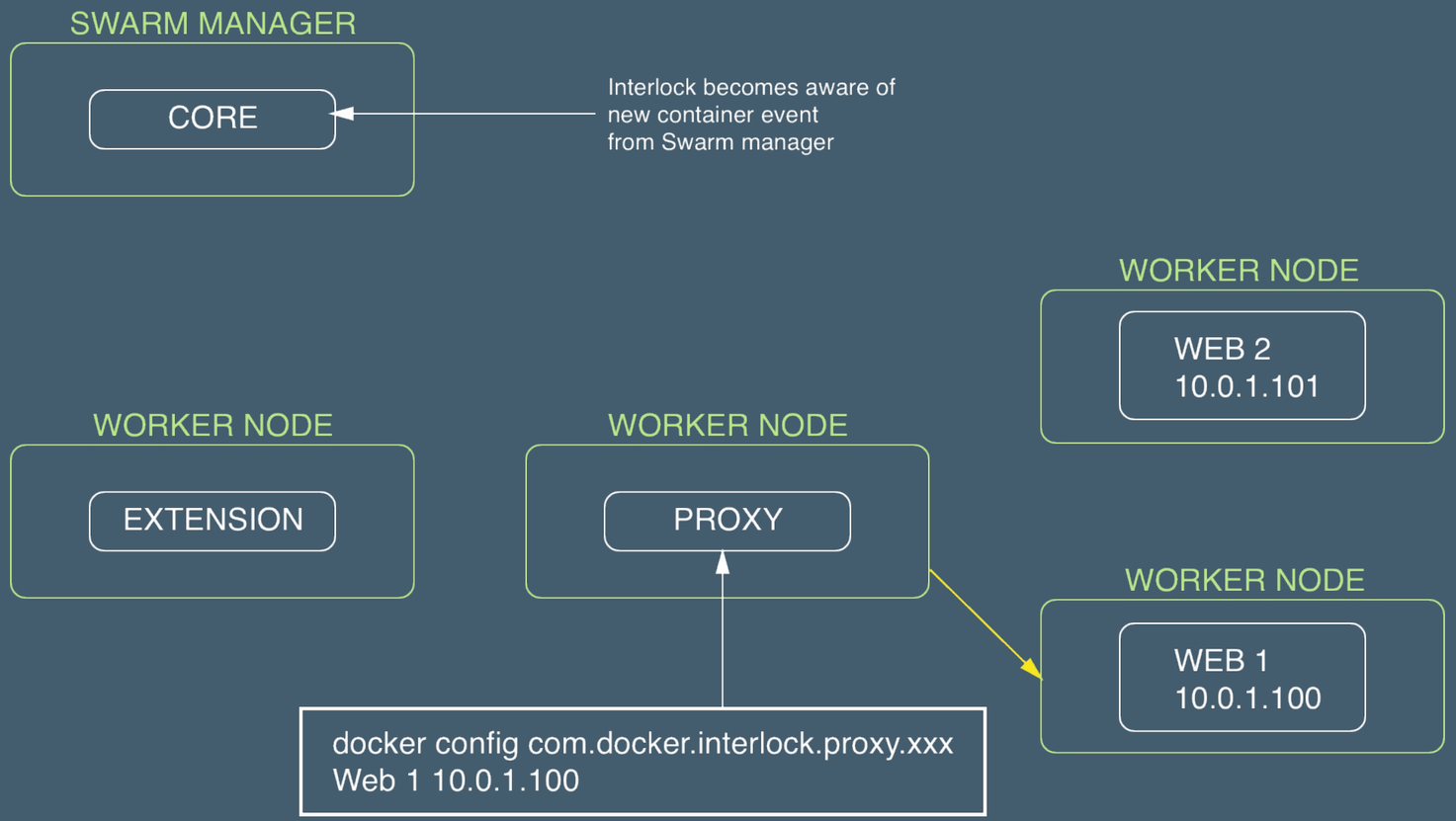
INTERLOCK 2 TRAFFIC



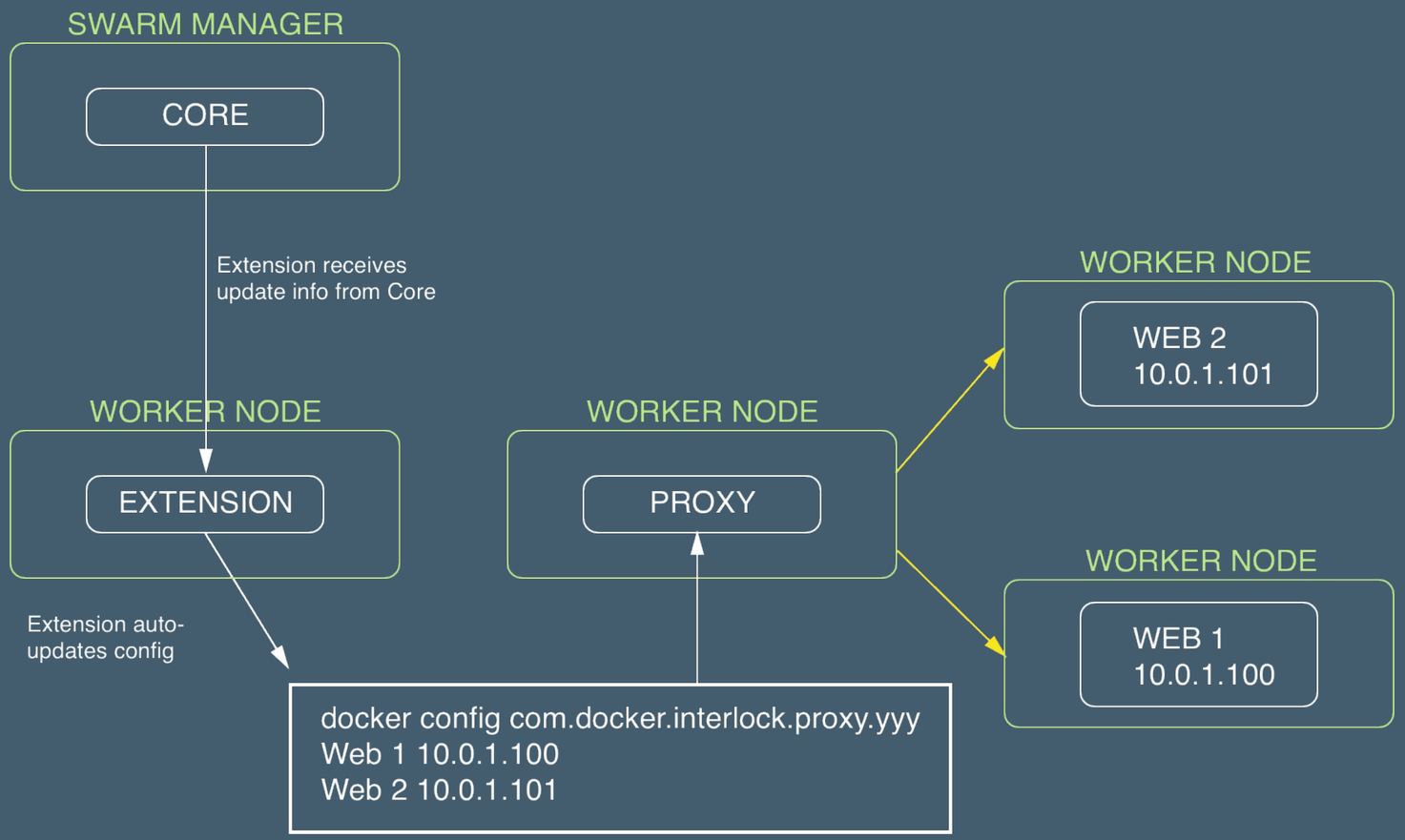
INTERLOCK 2 TRAFFIC



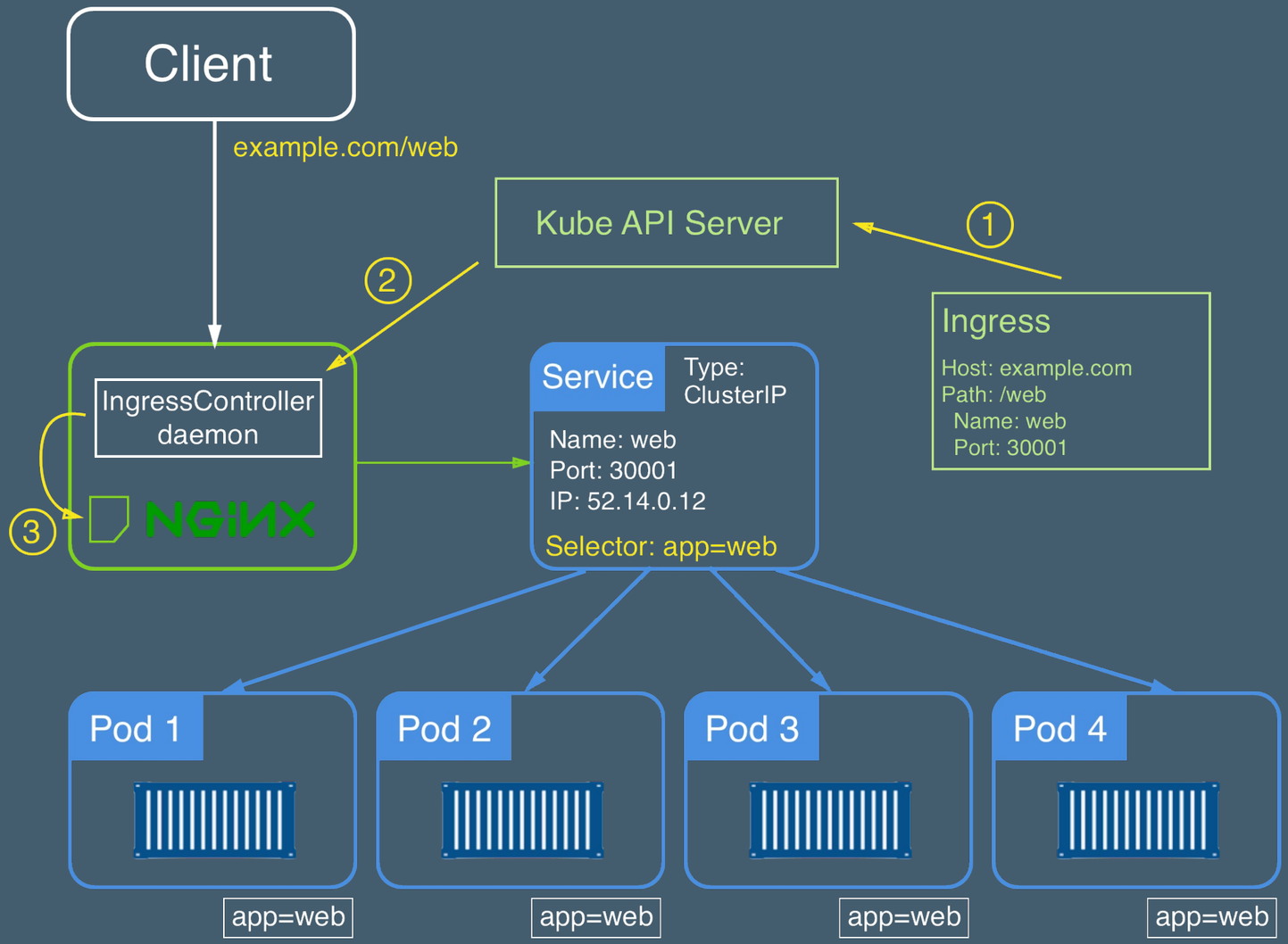
INTERLOCK 2 TRAFFIC



INTERLOCK 2 TRAFFIC



KUBERNETES INGRESS



NETWORKING SECURITY REMINDER

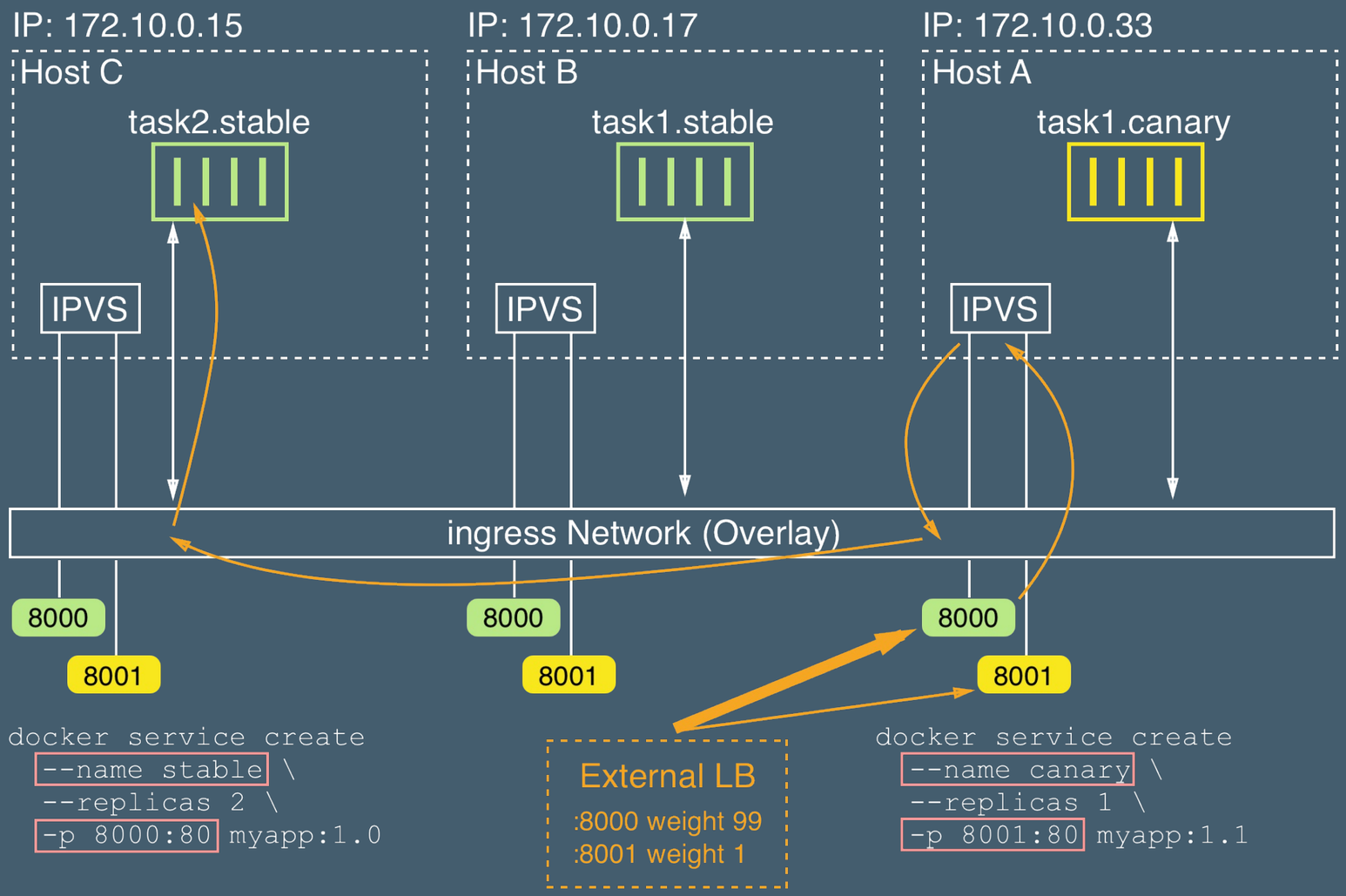
Always isolate containers that don't need to talk to each other.

- Swarm: separate software defined networks are mutually firewalled by default
- Kube: Impose a NetworkPolicy to isolate pod communication

NETWORKING-DEPENDENT DEPLOYMENT MODELS

- Canary releases
- Blue / green releases
- Both are essentially routing and load balancing problems
- Kube label selection very powerful for both

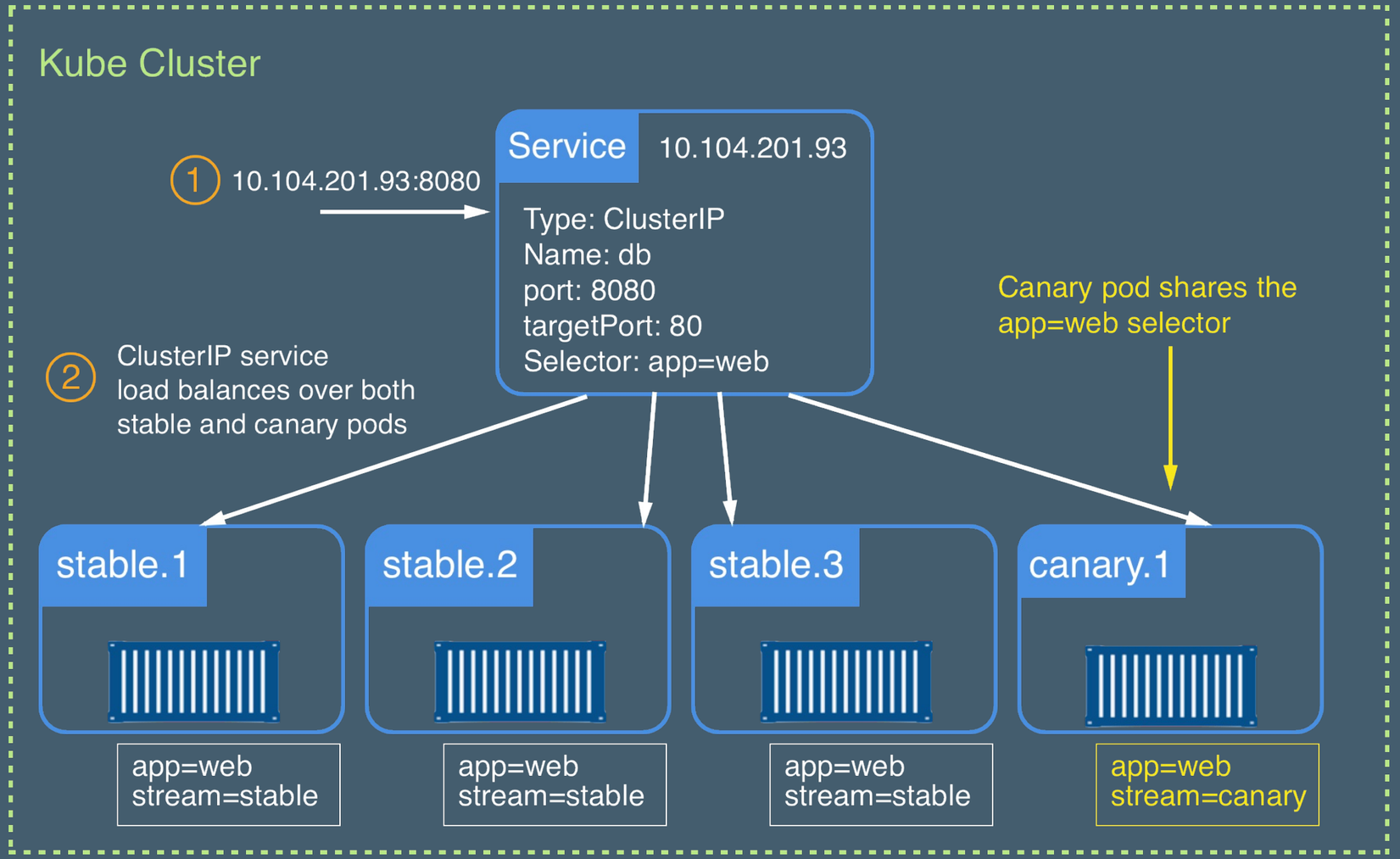
SWARM CANARY



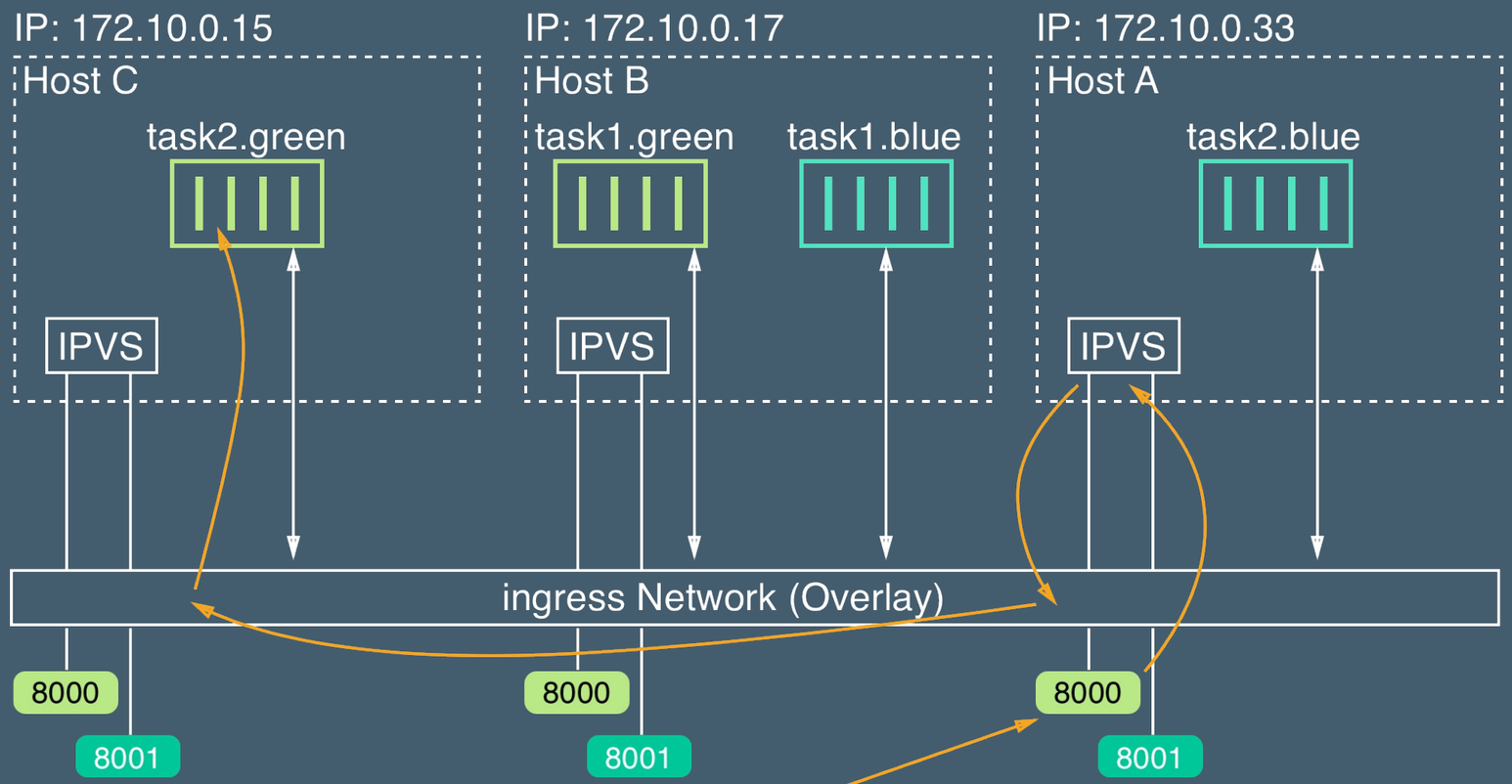

```
docker service create
  --name stable \
  --replicas 2 \
  -p 8000:80 myapp:1.0
```

```
docker service create
  --name canary \
  --replicas 1 \
  -p 8001:80 myapp:1.1
```

KUBE CANARY



SWARM BLUE / GREEN

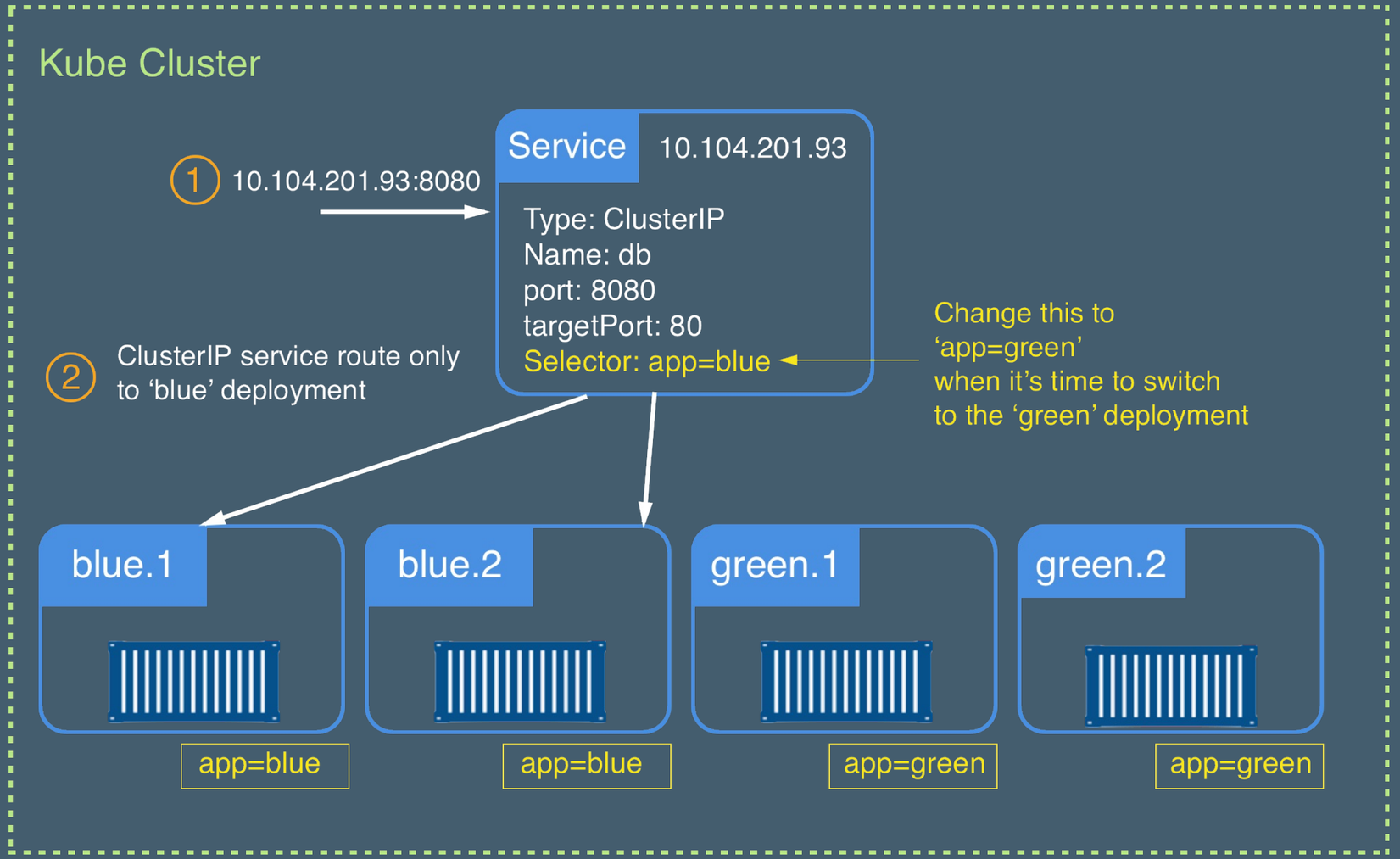



```
docker service create
  --name green \
  --replicas 2 \
  -p 8000:80 myapp:1.0
```

External LB

```
docker service create
  --name blue \
  --replicas 2 \
  -p 8001:80 myapp:1.1
```

KUBE BLUE / GREEN



EXERCISE: ADVANCED ROUTING MODELS

Work through:

- L7 Swarm Routing with Interlock
- Kubernetes Ingresses
- Release Models in Swarm
- Release Models in Kubernetes

in the Docker for Enterprise Operations Exercises book.

FURTHER READING

- Docker Networking Reference Arch.: https://dockr.ly/2q3O8jq
- UCP Load Balancing & Service Discovery: https://dockr.ly/2q4jVkY
- Kubernetes Services: https://bit.ly/2GSXwyB
- Managing Resources in Kube: https://bit.ly/2qIILVC
- Interlock for Production: https://dockr.ly/2R3v7JR
- Kubernetes Ingresses: https://bit.ly/2Nv8ose

LOGGING

DISCUSSION: LOGGING

What sort of logs and metrics do you want to collect from UCP?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Select and set a logging driver at the engine level, including `json-file` and `journald`
- Configure logging options
- Configure and retrieve UCP API audit logs
- Describe and implement a centralized logging solution

LOGGING DRIVERS

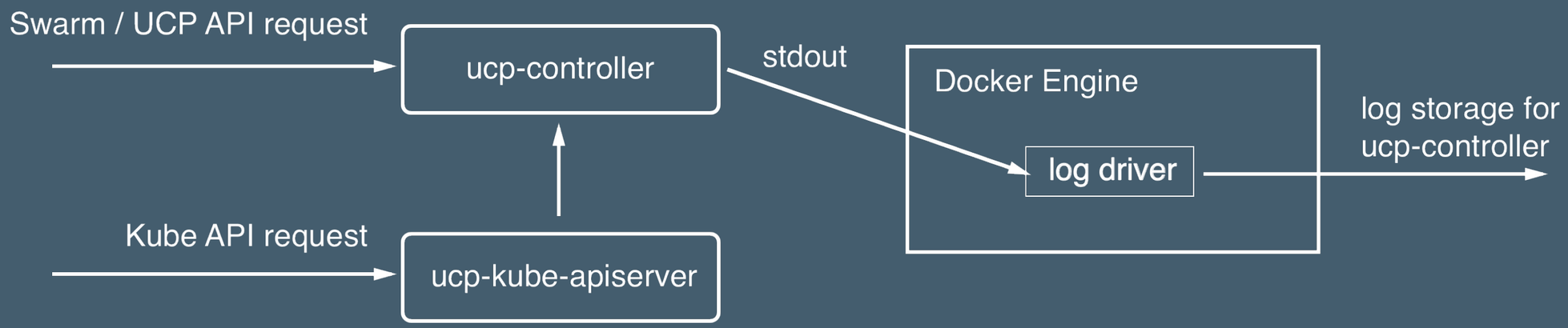
- Consume STDOUT + STDERR from PID 1
- Examples:
  - **json-file**: default, JSON formatted
  - **journald**: forward container logs to system journal
  - **local**: enables log reading for drivers that don't natively support reads
  - Many more: https://dockr.ly/2xuuukF
- Configured per engine in **daemon.json**

LOG COMPRESSION AND ROTATION

- Default: logs grow unbounded
- Set options in daemon.json to rotate log files and limit size
- local log driver compresses by default

AUDIT LOGS

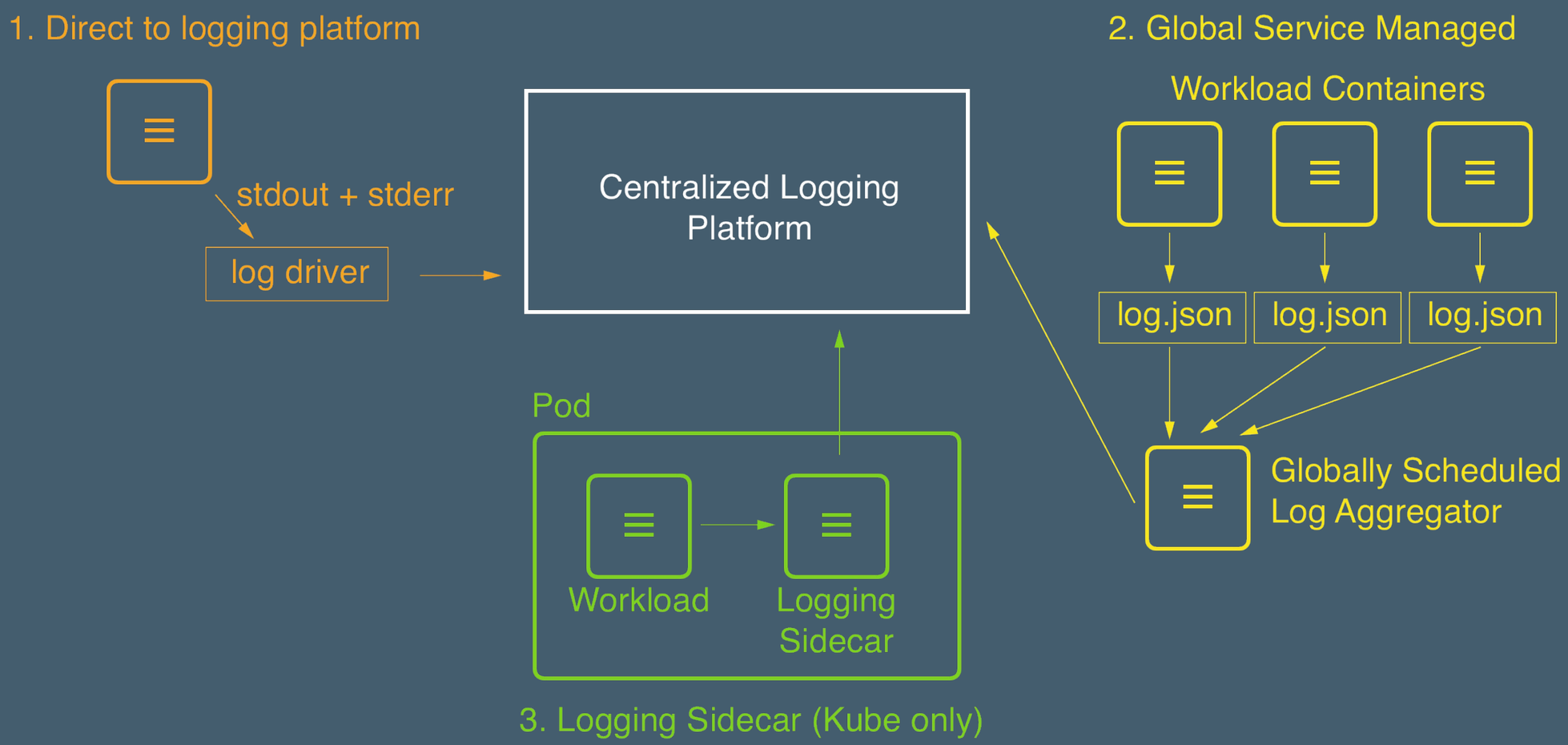
- Timestamp & user IDs invoking security-relevant UCP API calls
- Visible in container logs of ucp-controller



CENTRALIZED LOGGING

- Containerized deployments are distributed but connected
- large number of components = high rate of breakage
- Must correlate events...
- ... without granting access to production nodes.

CENTRALIZED LOGGING MODELS



EXERCISES: LOGGING

Work through:

- Configuring Engine Logs
- UCP Audit Logs
- Centralized Logging

in the Docker for Enterprise Operations Exercises book.

FURTHER READING

- View logs for a container or service: http://dockr.ly/2ezdZdI
- Docker Reference Architecture: Docker Logging Design and Best Practices: http://dockr.ly/2gG6ZjG

APPLICATION HEALTH & READINESS CHECKS

DISCUSSION: IS YOUR APPLICATION HEALTHY?

How do you determine if an application is healthy or not? What are some ways a process can become unhealthy without exiting?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

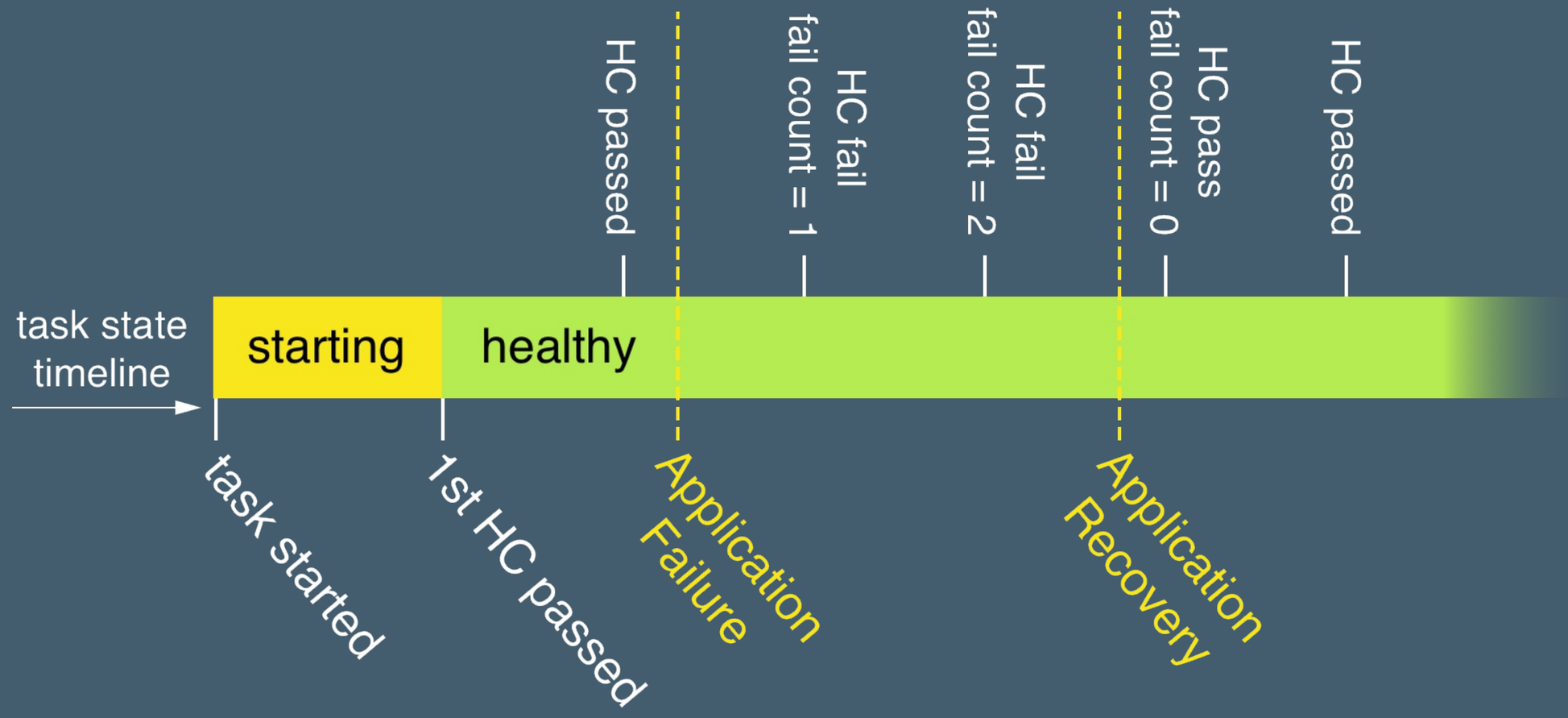
- Describe the steps of a generic health check protocol
- Configure Swarm and Kubernetes to kill unhealthy containers, and configure Kubernetes to remove unready pods from load balancing

HEALTH CHECK PROTOCOLS

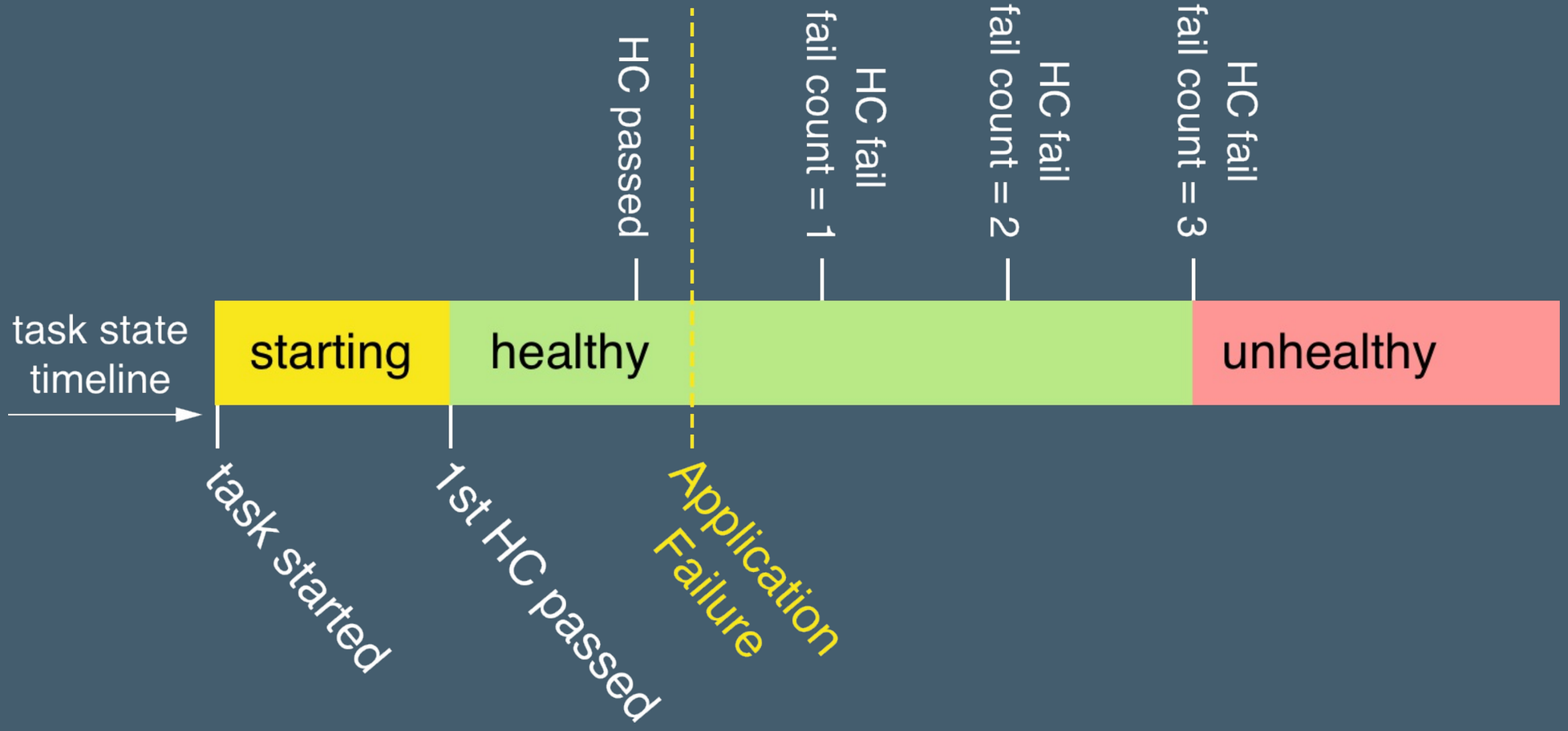
Monitoring application health requires:

- Action to check health
- Frequency of checks
- Timeout per check
- Number of checks

HEALTH CHECK PROTOCOL - HEALTHY



HEALTH CHECK PROTOCOL - UNHEALTHY



HEALTHCHECK: SWARM SERVICE CONTAINER

- Dockerfile

```
HEALTHCHECK CMD curl --fail http://localhost:5000/health || exit 1
```

- Docker Compose File

```
healthcheck:
    interval: 10s
    timeout: 2s
    retries: 3
    start-time: 30s
```

HEALTHCHECK: KUBE LIVENESS PROBE

Kube yaml:

```
kind: Pod
...
spec:
  containers:
  - name: demo
    image: ...
    livenessProbe:
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 3
      initialDelaySeconds: 30
      successThreshold: 1
      exec:
        command:
        - cat
        - /tmp/healthy
    ...
```

ALTERNATE LIVENESS PROBES

- HTTP Request: success if 200 <= response < 400
- TCP socket: success if connection succeeds on specified port

KUBERNETES READINESS PROBE

- Defined under readinessProbe
- Same syntax as livenessProbe
- No service traffic to pods w/ an unready container

EXERCISE: HEALTH CHECKS

Work through the 'Health Checks' exercise in your exercise book.

FURTHER READING

- Healthchecks in Dockerfiles: https://dockr.ly/2Su62tl
- Healthchecks in Compose files: https://dockr.ly/2yH2cVo
- Kube Liveness and Readiness Probes: https://bit.ly/2mauMH1

DOCKER TRUSTED REGISTRY

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Install DTR, configure its storage backend, and establish push and pull rights with a remote machine
- Overview a complete software supply chain supported by DTR
- Identify and troubleshoot common DTR installation problems

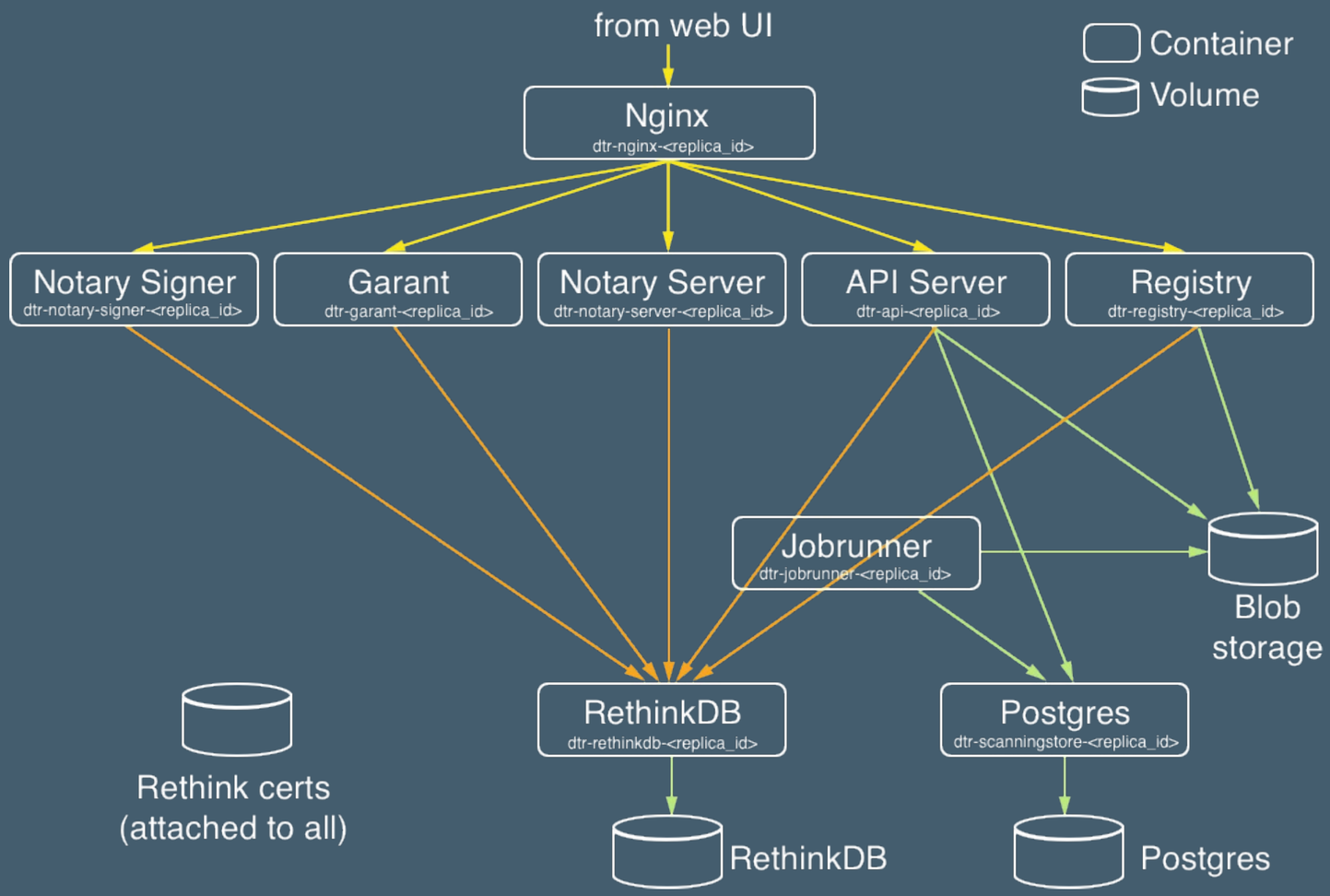
THE SOFTWARE SUPPLY CHAIN

- Image Creation
- Image Distribution
- Container Execution

DTR KEY FEATURES

- Image Creation:
  - Image Security Scanning
  - Repository Automation
- Image Distribution:
  - Content Trust / Notary
  - Content Cache
- Image Storage:
  - Pluggable Storage Drivers

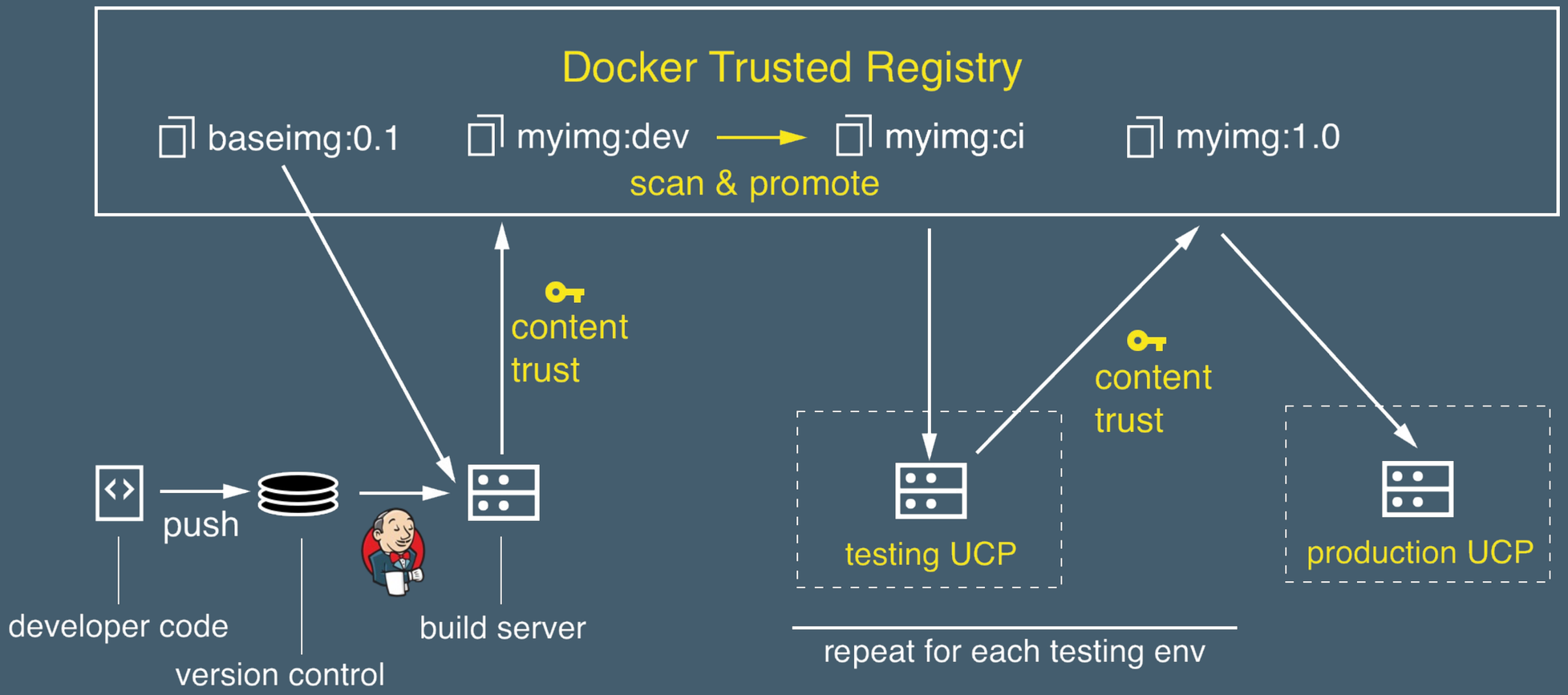
DTR ARCHITECTURE



DTR HIGH AVAILABILITY

- Extra DTR replicas provide HA for web app...
- ...not images!
- Need external storage (cloud, NFS) reachable by all DTR replicas

A DOCKER PIPELINE



DTR INSTALLATION

Work through:

- Installing Docker Trusted Registry
- Optional: Configuring DTR for High Availability
- Pushing and Pulling from DTR

in the Docker for Enterprise Operations Exercises book.

DTR INSTALLATION PROBLEMS

- "Failed to execute phase2" -> allow containers on managers
- x509 errors -> establish cert trust
- JWT expiration -> impose ntp clock sync

EMERGENCY REPAIR

- Allows an admin to recover a DTR cluster after a loss of quorum
- Works from a single dtr-rethink volume
- Use as a tool of last resort

FURTHER READING

- DTR architecture: https://dockr.ly/2JiEVvG

DTR ORGANIZATIONS AND TEAMS

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

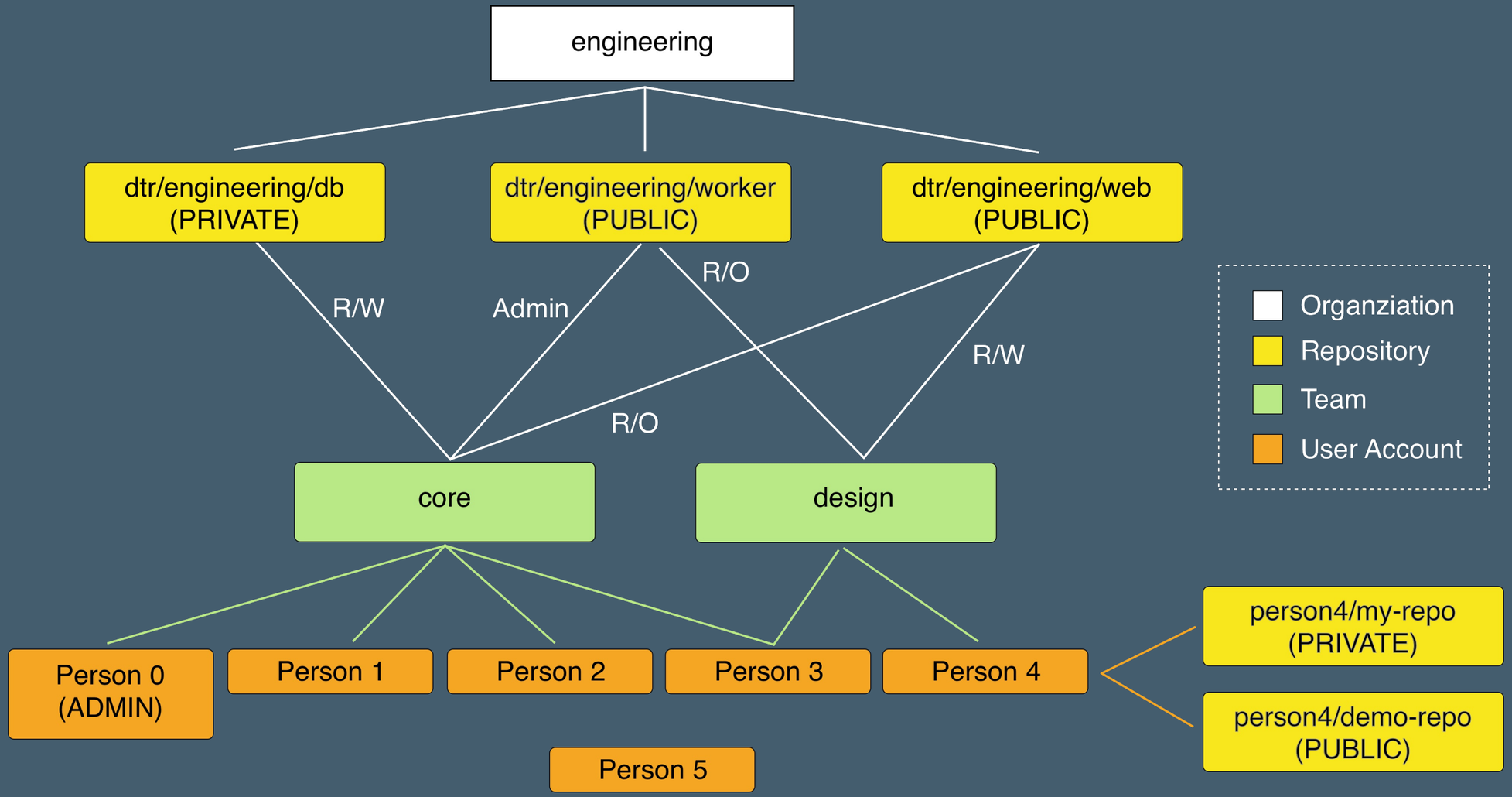
- Understand and manipulate role based access control in DTR

DTR ORG CHARTS

Four key entities:

- Organizations namespace all other assets.
- Repositories hold images.
- Teams define access control to repositories.
- Users are grouped by teams and orgs.

A 'SIMPLE' CASE



REPOSITORY PERMISSIONS

Repo access is controlled by two concerns:

- Public vs. Private, and Org vs User owned
- Team Permissions

PUBLIC/PRIVATE/OWNERSHIP MATRIX

	Public	Private
User-Owned	• Anyone can pull • Visible to all • Push by owner	• Only visible to owner & admins
Org-Owned	• Anyone can pull • Visible to all • Push by R/W team	• Must be R/W or R/O team to see repo

REPOSITORY PERMISSIONS

- Read only
  - Browse and pull
- Read write
  - Read only plus:
  - Push images
  - Delete tags
- Admin
  - Read write plus:
  - Edit repository description
  - Set public or private
  - Change team access level

ORGANIZATION MEMBERS

- Org member without team membership
- Team members are automatically organization members
- Organization members can:
  - View other members
  - View org teams and members
  - View and pull images from public repositories in the org
- No way to grant org member write access to org-owned repo.

ORGANIZATION OWNERS

- Individual org members can be made into an organization owner
- Organization owners:
  - Have admin access to all org-owned repos
  - Can manage org and team membership

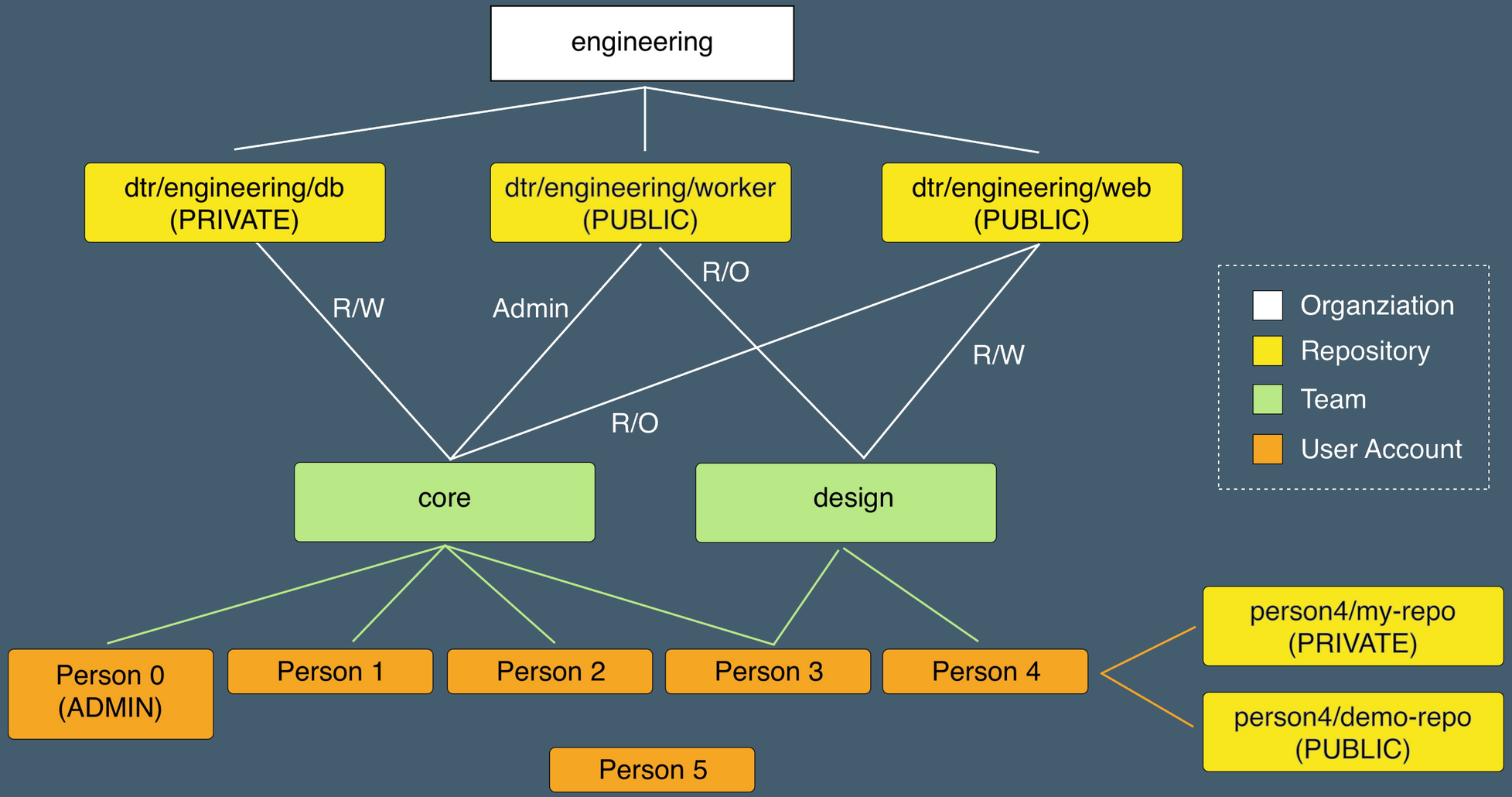
USER TOKEN MANAGEMENT

- Allows the management of 'docker login' tokens
- Avoids use of passwords

EXERCISE: DTR TEAMS

Work through the 'Working with Organizations and Teams' exercise in the Docker for Enterprise Operations exercise book.

QUIZ



FURTHER READING

- Create and Manage Organizations: https://dockr.ly/2qUoYVb

CONTENT TRUST

DISCUSSION: SOFTWARE APPROVAL PROCESS

Who has to sign off on a piece of software before it goes into production at your organization? How do you enforce this, and how do you ensure the sign-off hasn't been faked?

LEARNING OBJECTIVES

- Set up and manage content trust for a DTR repository
- Describe the differences in Docker's behavior with content trust enabled versus disabled
- Demand UCP only runs images signed by a set of teams

SHOULD WE RUN IT?

- Want a machine readable record of software approval process
- Want built in security to make sure sign-off is hard to fake

SOFTWARE DOWNLOADS

- Software downloads are not intrinsically secure:
  - Simple injection
  - Mix-and-match
  - Downgrade
- Man-in-the-middle countered by The Update Framework
- Generates signed metadata to certify software integrity and provenance
- Docker EE can use TUF metadata as a machine-readable signoff solution

THE UPDATE FRAMEWORK

4+1 keys to sign four pieces of metadata:

- Root: root of trust
- Target: signed hash of software
- Snapshot: signed valid combo of root cert + target metadata
- Timestamp: short-expiration signature on snapshot metadata
- (Optional) Delegation: Grant target signing authority to a third party

DOCKER NOTARY

- Implements TUF for Docker
- Server side: Notary integrated into DTR, handles Snapshot + Timestamp keys
- Client side: docker trust CLI, handles root, target, and delegation keys

CONTENT TRUST FOR IMAGE CONSUMERS

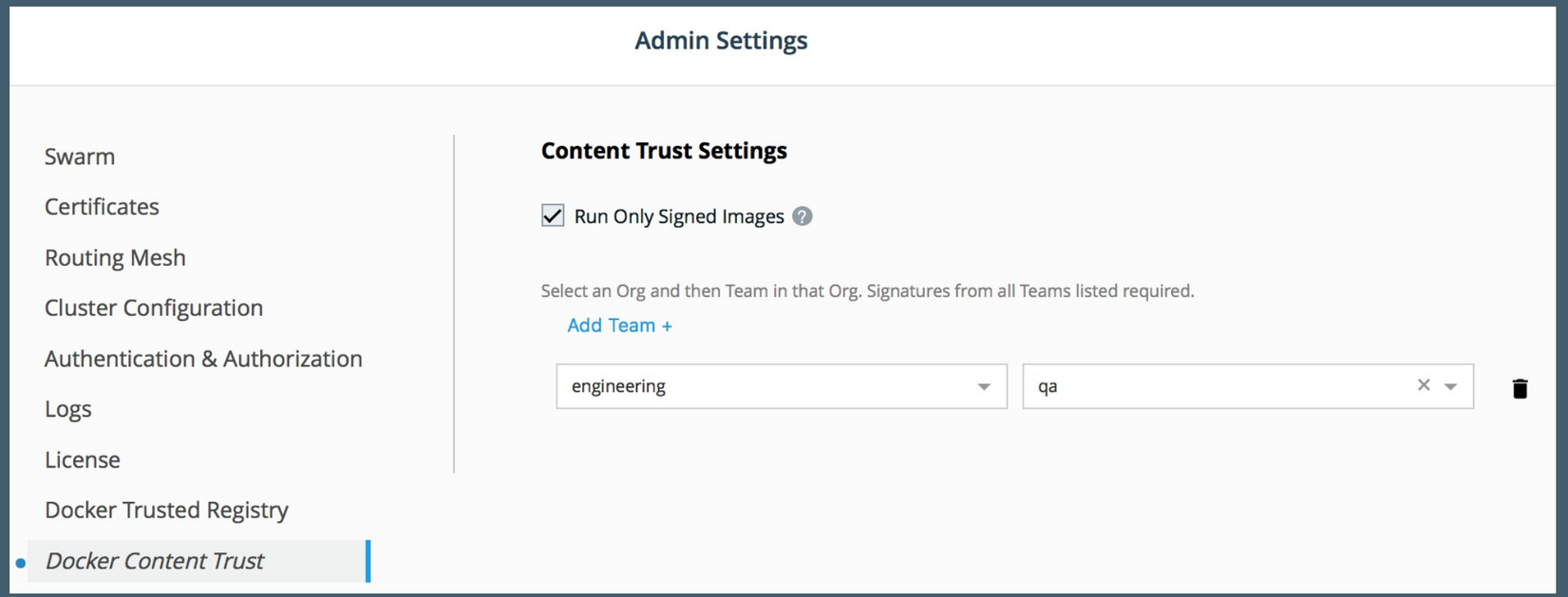
If content trust is enabled, only signed images are available for use with:

- docker image push
- docker image pull
- docker image build
- docker container create
- docker container run
- docker service create

(But can be circumvented with --disable-content-trust in most cases)

INTEGRATION WITH UCP

- UCP can be configured to only run signed images
- Checked during application deployment
- Any unsigned image will be rejected
- Multiple signatures possible



TRUST PINNING

- Configure Docker Engine to only accept whitelisted root signers
- Configured through daemon.json
- Not circumvented by --disable-content-trust

EXERCISE: CONTENT TRUST

Work through the 'Content Trust' exercise in your exercise book.

FURTHER READING

- Intro to content trust: http://dockr.ly/2gAgIo3
- Features of Docker content trust: http://dockr.ly/1EyCxrR
- Automation with content trust: http://dockr.ly/2x5lLaC

IMAGE SECURITY SCANNING

DISCUSSION

- How do we know what's in an image?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe the process DTR takes to scan images for vulnerabilities
- Set up security scanning on DTR, and configure it for an individual repository
- Use DTR’s UI to find out an image's vulnerabilities, and more information about that vulnerability

WHAT'S IN YOUR IMAGES?

- Components installed in base layer
- Components installed in custom layers
- Known vulnerabilities?
- (Currently) unknown vulnerabilities?

DTR SECURITY SCANNING FLOW

1. New layer pushed to DTR
2. Scan triggered (auto or manual)
3. Bill-of-materials generated and saved (slow, resource intensive)
4. BoM re-checked against CVE database every time the db is updated (fast)
5. Regular database updates from https://dss-cve-updates.docker.com/

INSTRUCTOR DEMO: SECURITY SCANNING

See the 'Security Scanning' demo in the Docker for Enterprise Operations Exercises book.

COMMON SCANNING MISTAKES

- Make sure initial CVE database is downloaded before starting scans
- Make sure all DTR replicas can reach storage backend
- Consider manual-only scans as part of pipeline

FURTHER READING

- Set up security scanning in DTR: https://dockr.ly/2HMgp9d
- Scan images for vulnerabilities: https://dockr.ly/2HNOdCK

REPOSITORY AUTOMATION

DISCUSSION: SOFTWARE AUTOMATION

What automation do you currently use in your software development cycle? How will that connect with Docker?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Automatically retag an image from one DTR repo to another
- Define webhooks triggered by DTR events
- Integrate DTR into a CI/CD chain using the above

AUTOMATION TOOLS

- Image Promotion & Mirroring
- Webhooks

IMAGE PROMOTION

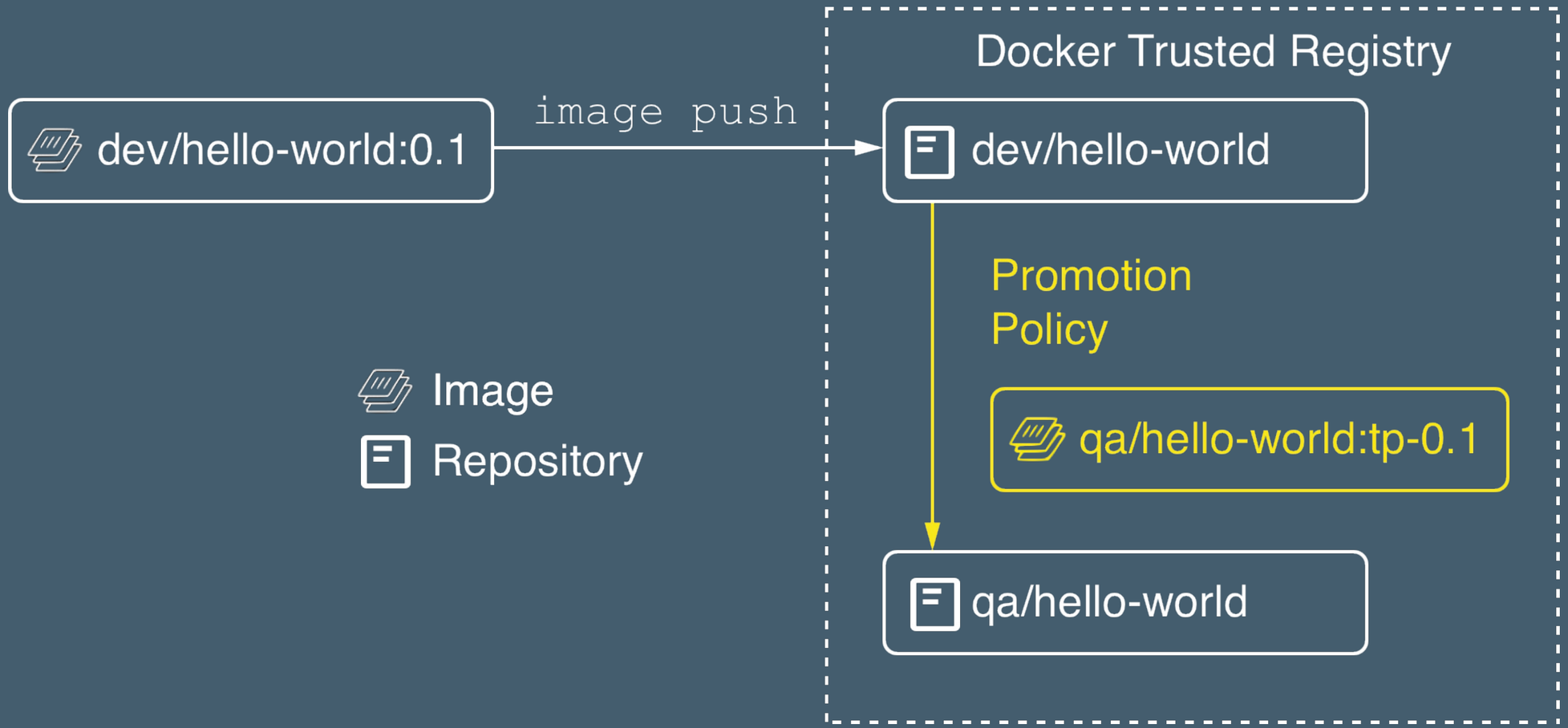


IMAGE PROMOTION POLICIES

- Manual or automatic
- Automatic promotion can be triggered on:
  - Tag name
  - Package name
  - Minor / Major / Critical / All vulnerabilities
  - Component licenses
- No limit to number of policies that can be defined

IMAGE MIRRORING

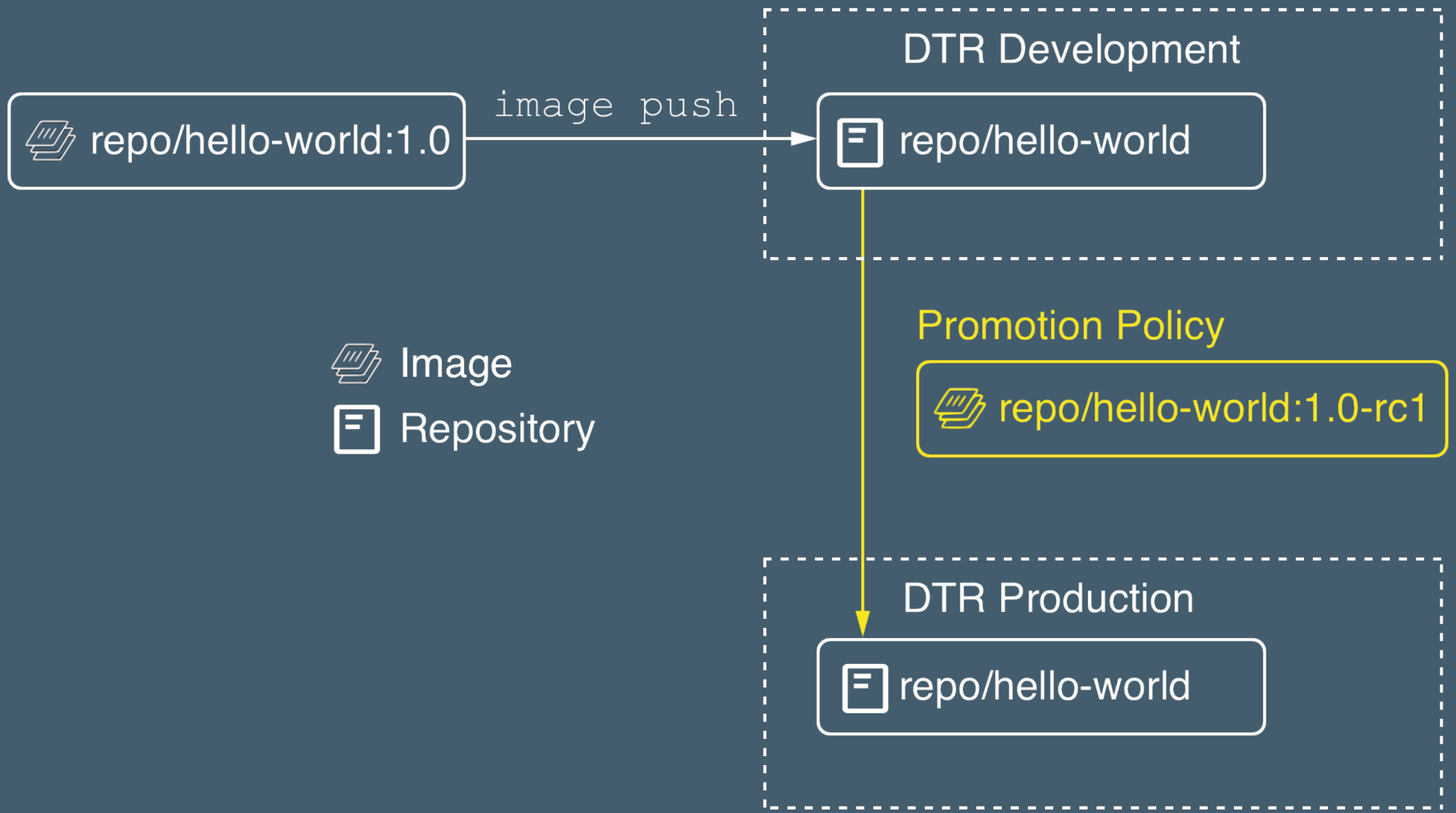
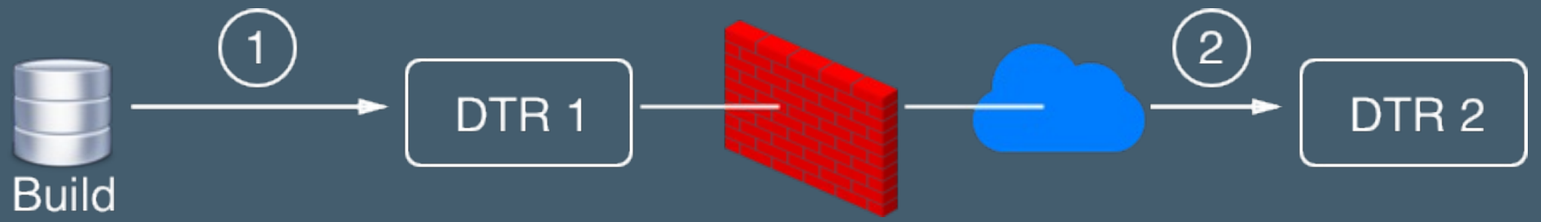


IMAGE MIRRORING - PUSH BASED



- Image pushed to DTR 1
- If policies met => push to DTR 2
- AuthN & AuthZ managed by each DTR
- Signing & scan data not (yet) preserved

IMAGE MIRRORING - PULL BASED

- Image pushed to DTR 1
- DTR 2 polls DTR 1 for updates
- New image found => pull to DTR 2
- Combine with Promotion Policies

WEBHOOKS

POST message with JSON payload, triggered on:

- Tag push or delete
- Manifest push or delete
- Security scan failed
- Security scan complete

Defined per repository.

WEBHOOK PAYLOAD

- Webhook payloads always come in a wrapper:

```
{
 "type": "...",
 "createdAt": "2012-04-23T18:25:43.511Z",
 "contents": {...}
}
```

- The contents key depends on the event type; see https://dockr.ly/2JjcAW7 for the full spec.

AUDITING DTR ACTIONS

- Per repo: Activity monitor (ex: push, delete, scan, promote)
- All DTR: Job log (ex: scanning, garbage collection, webhooks, pruning)

EXERCISE: REPOSITORY AUTOMATION

Work through the 'Image Promotion & Webhooks' exercise in your exercise book.

DISCUSSION

- What are the pros and cons of promoting images in a single DTR, versus mirroring them across multiple DTRs?

FURTHER READING

- Managing webhooks: https://dockr.ly/2JjcAW7
- Promotion policies overview: https://dockr.ly/2KarnDN
- Image Promotions and Immutable Repos: http://bit.ly/2eEz7TH

IMAGE MANAGEMENT

DISCUSSION: IMAGE RETENTION

How long should you keep your images around for? What requirements affect this decision?

LEARNING OBJECTIVES

- Configure content caching, tag pruning and garbage collection in DTR
- Design and provision a DTR deployment appropriate for development and production use cases
- Plan DTR deployments appropriate to an image retention policy

IMAGE RETENTION

- Must satisfy audit and rollback requirements
- Decisions affect DTR resourcing plan; figure this out before deploying to production!
- Clear deletion policy allows automation
- Need to purge old tags, and clean up disk

IMAGE MANAGEMENT AUTOMATION

- Tag Pruning: policy-based purging of unneeded tags
- Garbage Collection: delete unreferenced image layers from the storage backend

TAG PRUNING

- Problem: A lot of images are created or retagged, and many of them are not used anymore
- Tag pruning automates tag deletion

SETTING UP TAG PRUNING

DTR offers two main options:

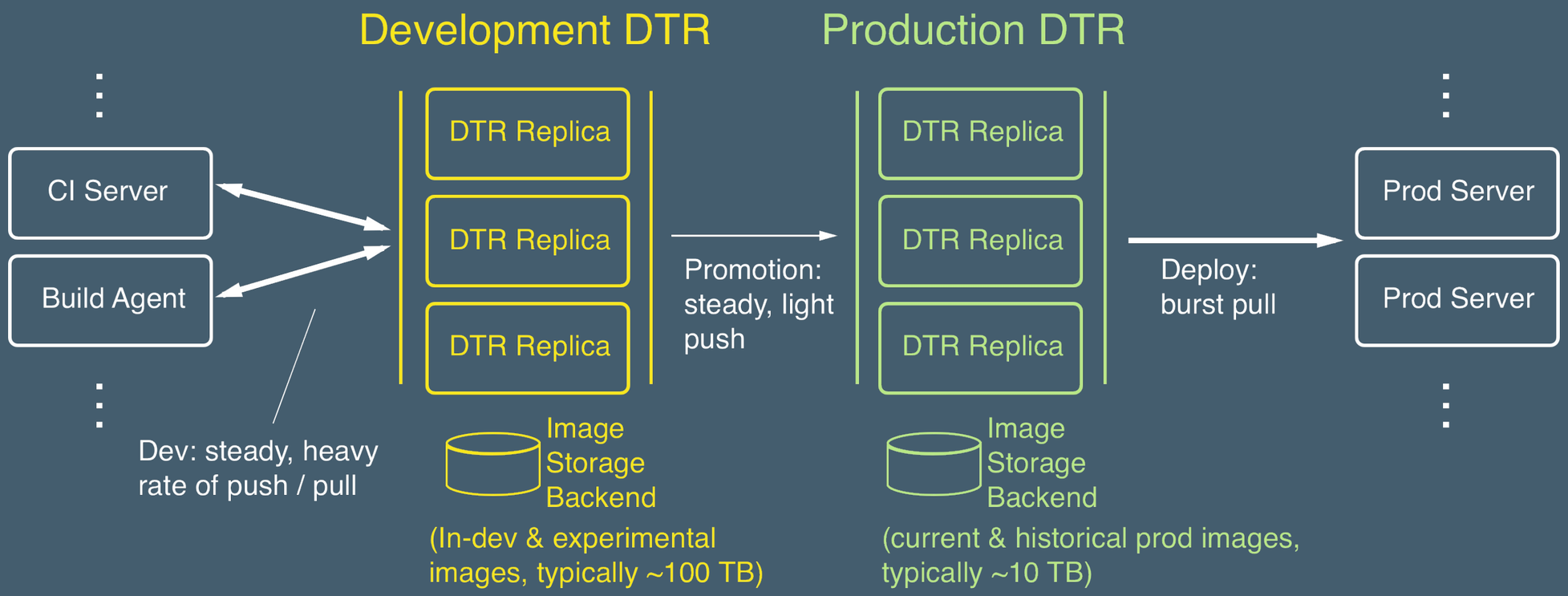
1. Set a maximum number per repo
2. Define a policy (by time, vulnerability, etc.)

Note: Be careful not to prune images still in production!

GARBAGE COLLECTION

- Disabled by default
- Scheduled as cronjob
- Resource intensive; consider time-limiting

DTR ENVIRONMENT ISOLATION



CONTENT CACHING

- Geographically distributed image caches
- Configurable per DTR user
- Appropriate for reducing download latency, DTR load

EXERCISE: IMAGE MANAGEMENT

Work through:

- Tag Pruning & Garbage Collection
- Optional: Content Caching

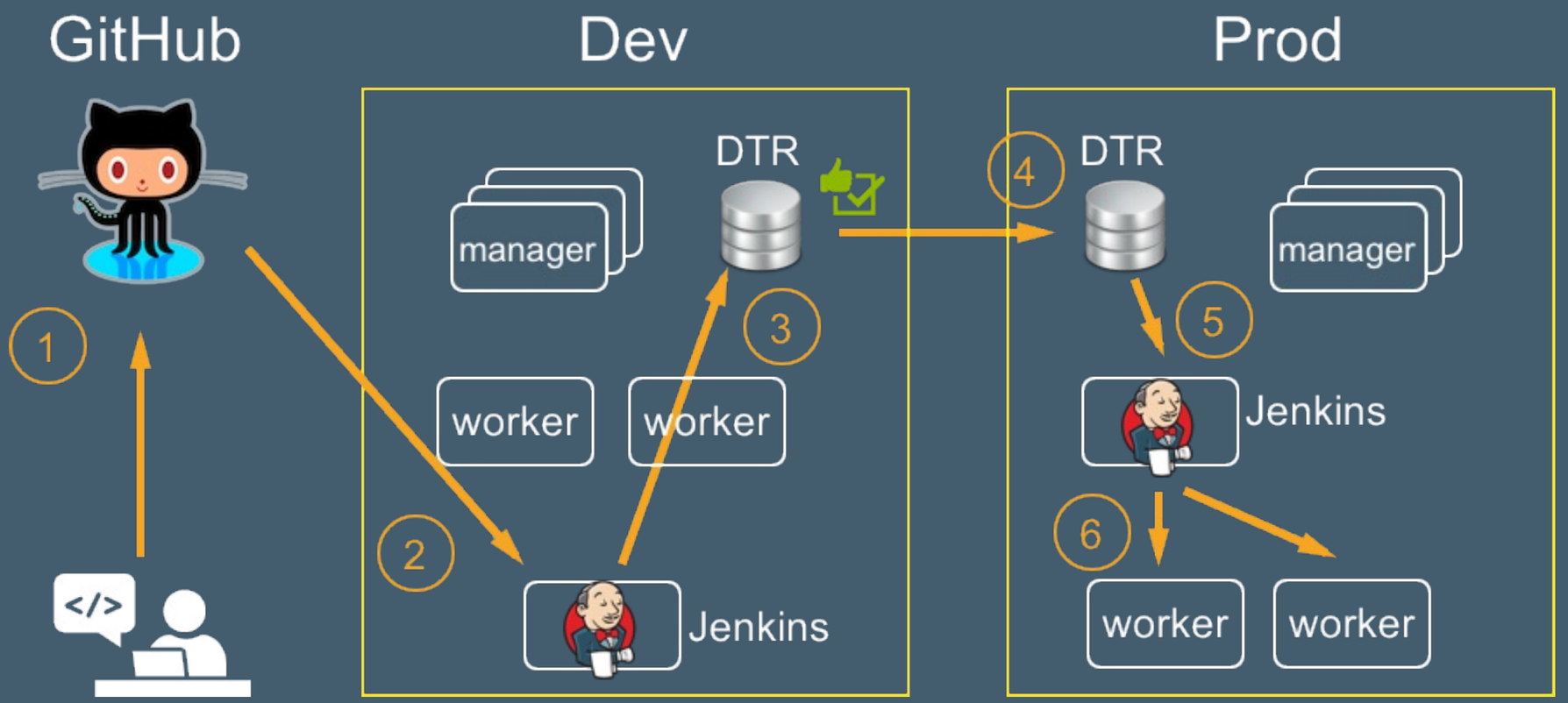
in your exercise book.

FURTHER READING

- Garbage Collection: https://dockr.ly/2EdcXnt
- DTR Content Caches: https://dockr.ly/2yq5UBW

OPERATIONS SIGNATURE ASSIGNMENT

THE USE CASE


GitHub
Dev
Prod
DTR
manager
worker
worker
Jenkins
1
2
3
4
DTR
manager
5
Jenkins
6
worker
worker
</>

SIGNATURE ASSIGNMENT: THE SOFTWARE SUPPLY CHAIN

Work through the 'The Software Supply Chain' exercise in your exercise book.

DOCKER FOR ENTERPRISE OPERATIONS

Thanks for coming! Please take one of our feedback surveys:

- Docker for Enterprise Operations (standalone): http://bit.ly/2FiYZ0b
- Docker Fundamentals + Enterprise Ops (combined class): https://bit.ly/2J1ryiT

Get in touch: training@docker.com

success.docker.com/training

YOU'RE ON YOUR WAY TO BECOMING DOCKER CERTIFIED!

- Study up with our Study Guides at http://bit.ly/2yPzAdb
- Take it online 24 hours a day
- Results delivered immediately
- Benefits include:
  - Digital certificate
  - Online verification
  - Private LinkedIn group
  - Exclusive events

SUCCESS.DOCKER.COM/CERTIFICATION